Category Theory for ZFC in HOL III
Universal Constructions for 1-Categories

Mihails Milehins

March 19, 2025

Abstract

This article provides a formalization of elements of the theory of universal constructions for 1-categories (such as limits, adjoints and Kan extensions) in the object logic *ZFC in HOL* ([13], also see [11]) of the formal proof assistant *Isabelle* [12].

Acknowledgements

The author would like to acknowledge the assistance that he received from the users of the mailing list of Isabelle in the form of answers given to his general queries. Special thanks go to Andreas Lochbihler for suggesting the use of the combination of unrestricted overloading and locales for structuring mathematical knowledge on the mailing list of Isabelle: the design pattern that is used in this study builds upon this idea. Special thanks also go to Thomas Sewell for suggesting a trick for rewriting expressions modulo associativity on the mailing list of Isabelle, which was used on numerous occasions throughout the development of this work. Furthermore, the author would like to mention that the tool "Sketch-and-Explore" [5] from the standard distribution of Isabelle was used extensively in the development of this work. Moreover, the author would like to acknowledge the positive role that numerous Q&A posted on the Stack Exchange network (especially Mathematics Stack Exchange, Stack Overflow and TeX Stack Exchange) played in the development of this work. Lastly, the author would like to express gratitude to all members of his family and friends for their continuous support.

Contents

1 Introduction

This article provides a formalization of further elements of the theory of 1-categories without an additional structure. More specifically, this article explores canonical universal constructions [1][1] and their properties, building upon the formalization of the foundations of category theory in [10].

[1] https://ncatlab.org/nlab/show/universal+construction

2 Universal arrow

2.1 Background

The following section is based, primarily, on the elements of the content of Chapter III-1 in [9].

named-theorems *ua-field-simps*

definition *UObj* :: *V* **where** [*ua-field-simps*]: *UObj* = *0*
definition *UArr* :: *V* **where** [*ua-field-simps*]: *UArr* = $1_{\mathbb{N}}$

lemma [*cat-cs-simps*]:
  shows *UObj-simp*: $[a, b]_{\circ}(\!|UObj|\!) = a$
    and *UArr-simp*: $[a, b]_{\circ}(\!|UArr|\!) = b$
  unfolding *ua-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

2.2 Universal map

The universal map is a convenience utility that allows treating a part of the definition of the universal arrow as an arrow in the category *Set*.

2.2.1 Definition and elementary properties

definition *umap-of* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *umap-of* $\mathfrak{F}$ *c r u d* =
    [
      $(\lambda f' \in_{\circ} Hom\ (\mathfrak{F}(\!|HomDom|\!))\ r\ d.\ \mathfrak{F}(\!|ArrMap|\!)(\!|f'|\!) \circ_{A} \mathfrak{F}(\!|HomCod|\!)\ u)$,
      $Hom\ (\mathfrak{F}(\!|HomDom|\!))\ r\ d$,
      $Hom\ (\mathfrak{F}(\!|HomCod|\!))\ c\ (\mathfrak{F}(\!|ObjMap|\!)(\!|d|\!))$
    ]$_{\circ}$

definition *umap-fo* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *umap-fo* $\mathfrak{F}$ *c r u d* = *umap-of* (*op-cf* $\mathfrak{F}$) *c r u d*

Components.

lemma (**in** *is-functor*) *umap-of-components*:
  assumes $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
  shows *umap-of* $\mathfrak{F}$ *c r u d*$(\!|ArrVal|\!)$ = $(\lambda f' \in_{\circ} Hom\ \mathfrak{A}\ r\ d.\ \mathfrak{F}(\!|ArrMap|\!)(\!|f'|\!) \circ_{A\mathfrak{B}} u)$
    and *umap-of* $\mathfrak{F}$ *c r u d*$(\!|ArrDom|\!)$ = *Hom* $\mathfrak{A}$ *r d*
    and *umap-of* $\mathfrak{F}$ *c r u d*$(\!|ArrCod|\!)$ = *Hom* $\mathfrak{B}$ *c* $(\mathfrak{F}(\!|ObjMap|\!)(\!|d|\!))$
  unfolding *umap-of-def arr-field-simps*
  by (*simp-all add*: *cat-cs-simps nat-omega-simps*)

lemma (**in** *is-functor*) *umap-fo-components*:
  assumes $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
  shows *umap-fo* $\mathfrak{F}$ *c r u d*$(\!|ArrVal|\!)$ = $(\lambda f' \in_{\circ} Hom\ \mathfrak{A}\ d\ r.\ u \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f'|\!))$
    and *umap-fo* $\mathfrak{F}$ *c r u d*$(\!|ArrDom|\!)$ = *Hom* $\mathfrak{A}$ *d r*
    and *umap-fo* $\mathfrak{F}$ *c r u d*$(\!|ArrCod|\!)$ = *Hom* $\mathfrak{B}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|d|\!))$ *c*
  unfolding
    umap-fo-def
    is-functor.umap-of-components[
      OF is-functor-op, *unfolded cat-op-simps*, *OF assms*
    ]
proof(*rule vsv-eqI*)
  fix f' **assume** $f' \in_{\circ} \mathcal{D}_{\circ}\ (\lambda f' \in_{\circ} Hom\ \mathfrak{A}\ d\ r.\ \mathfrak{F}(\!|ArrMap|\!)(\!|f'|\!) \circ_{A\ op\text{-}cat\ \mathfrak{B}}\ u)$
  then have f': $f' : d \mapsto_{\mathfrak{A}} r$ **by** *simp*
  then have $\mathfrak{F}f'$: $\mathfrak{F}(\!|ArrMap|\!)(\!|f'|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|d|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
    by (*auto intro*: *cat-cs-intros*)

from f' **show**
$(\lambda f' \in_\circ Hom\ \mathfrak{A}\ d\ r.\ \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor) \circ_{A\ op\text{-}cat\ \mathfrak{B}}\ u)(\lfloor f' \rfloor) =$
$(\lambda f' \in_\circ Hom\ \mathfrak{A}\ d\ r.\ u \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor))(\lfloor f' \rfloor)$
by (*simp add*: *HomCod.op-cat-Comp*[*OF assms* $\mathfrak{F}f'$])
qed *simp-all*

Universal maps for the opposite functor.

lemma (**in** *is-functor*) *op-umap-of*[*cat-op-simps*]: *umap-of* (*op-cf* $\mathfrak{F}$) = *umap-fo* $\mathfrak{F}$
unfolding *umap-fo-def* **by** *simp*

lemma (**in** *is-functor*) *op-umap-fo*[*cat-op-simps*]: *umap-fo* (*op-cf* $\mathfrak{F}$) = *umap-of* $\mathfrak{F}$
unfolding *umap-fo-def* **by** (*simp add*: *cat-op-simps*)

lemmas [*cat-op-simps*] =
is-functor.op-umap-of
is-functor.op-umap-fo

2.2.2 Arrow value

lemma *umap-of-ArrVal-vsv*[*cat-cs-intros*]: *vsv* (*umap-of* $\mathfrak{F}$ *c r u d*($\lfloor ArrVal \rfloor$))
unfolding *umap-of-def arr-field-simps* **by** (*simp add*: *nat-omega-simps*)

lemma *umap-fo-ArrVal-vsv*[*cat-cs-intros*]: *vsv* (*umap-fo* $\mathfrak{F}$ *c r u d*($\lfloor ArrVal \rfloor$))
unfolding *umap-fo-def* **by** (*rule umap-of-ArrVal-vsv*)

lemma (**in** *is-functor*) *umap-of-ArrVal-vdomain*:
assumes $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$
shows $\mathcal{D}_\circ$ (*umap-of* $\mathfrak{F}$ *c r u d*($\lfloor ArrVal \rfloor$)) = *Hom* $\mathfrak{A}$ *r d*
unfolding *umap-of-components*[*OF assms*] **by** *simp*

lemmas [*cat-cs-simps*] = *is-functor.umap-of-ArrVal-vdomain*

lemma (**in** *is-functor*) *umap-fo-ArrVal-vdomain*:
assumes $u : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor) \mapsto_{\mathfrak{B}} c$
shows $\mathcal{D}_\circ$ (*umap-fo* $\mathfrak{F}$ *c r u d*($\lfloor ArrVal \rfloor$)) = *Hom* $\mathfrak{A}$ *d r*
unfolding *umap-fo-components*[*OF assms*] **by** *simp*

lemmas [*cat-cs-simps*] = *is-functor.umap-fo-ArrVal-vdomain*

lemma (**in** *is-functor*) *umap-of-ArrVal-app*:
assumes $f' : r \mapsto_{\mathfrak{A}} d$ **and** $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$
shows *umap-of* $\mathfrak{F}$ *c r u d*($\lfloor ArrVal \rfloor$)($\lfloor f' \rfloor$) = $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor) \circ_{A\mathfrak{B}} u$
using *assms*(*1*) **unfolding** *umap-of-components*[*OF assms*(*2*)] **by** *simp*

lemmas [*cat-cs-simps*] = *is-functor.umap-of-ArrVal-app*

lemma (**in** *is-functor*) *umap-fo-ArrVal-app*:
assumes $f' : d \mapsto_{\mathfrak{A}} r$ **and** $u : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor) \mapsto_{\mathfrak{B}} c$
shows *umap-fo* $\mathfrak{F}$ *c r u d*($\lfloor ArrVal \rfloor$)($\lfloor f' \rfloor$) = $u \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor)$
proof–
from *assms* **have** $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor d \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$
by (*auto intro*: *cat-cs-intros*)
from *this assms*(*2*) **have** $\mathfrak{F}f'$[*simp*]:
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor) \circ_{A\ op\text{-}cat\ \mathfrak{B}}\ u = u \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f' \rfloor)$
by (*simp add*: *cat-op-simps*)
from
is-functor-axioms
is-functor.umap-of-ArrVal-app[

```
    OF is-functor-op, unfolded cat-op-simps,
    OF assms
    ]
  show ?thesis
    by (simp add: cat-op-simps cat-cs-simps)
qed

lemmas [cat-cs-simps] = is-functor.umap-fo-ArrVal-app

lemma (in is-functor) umap-of-ArrVal-vrange:
  assumes u : c ↦_𝔅 𝔉(|ObjMap|)(|r|)
  shows ℛ∘ (umap-of 𝔉 c r u d(|ArrVal|)) ⊆∘ Hom 𝔅 c (𝔉(|ObjMap|)(|d|))
proof(intro vsubset-antisym vsubsetI)
  interpret vsv ‹umap-of 𝔉 c r u d(|ArrVal|)›
    unfolding umap-of-components[OF assms] by simp
  fix g assume g ∈∘ ℛ∘ (umap-of 𝔉 c r u d(|ArrVal|))
  then obtain f'
    where g-def: g = umap-of 𝔉 c r u d(|ArrVal|)(|f'|)
      and f': f' ∈∘ 𝒟∘ (umap-of 𝔉 c r u d(|ArrVal|))
    unfolding umap-of-components[OF assms] by auto
  then have f': f' : r ↦_𝔄 d
    unfolding umap-of-ArrVal-vdomain[OF assms] by simp
  then have 𝔉f': 𝔉(|ArrMap|)(|f'|) : 𝔉(|ObjMap|)(|r|) ↦_𝔅 𝔉(|ObjMap|)(|d|)
    by (auto intro!: cat-cs-intros)
  have g-def: g = 𝔉(|ArrMap|)(|f'|) ∘_A𝔅 u
    unfolding g-def umap-of-ArrVal-app[OF f' assms]..
  from 𝔉f' assms show g ∈∘ Hom 𝔅 c (𝔉(|ObjMap|)(|d|))
    unfolding g-def by (auto intro: cat-cs-intros)
qed

lemma (in is-functor) umap-fo-ArrVal-vrange:
  assumes u : 𝔉(|ObjMap|)(|r|) ↦_𝔅 c
  shows ℛ∘ (umap-fo 𝔉 c r u d(|ArrVal|)) ⊆∘ Hom 𝔅 (𝔉(|ObjMap|)(|d|)) c
  by
    (
      rule is-functor.umap-of-ArrVal-vrange[
        OF is-functor-op, unfolded cat-op-simps, OF assms, folded umap-fo-def
        ]
    )
```

2.2.3 Universal map is an arrow in the category *Set*

```
lemma (in is-functor) cf-arr-Set-umap-of:
  assumes category α 𝔄
    and category α 𝔅
    and r: r ∈∘ 𝔄(|Obj|)
    and d: d ∈∘ 𝔄(|Obj|)
    and u: u : c ↦_𝔅 𝔉(|ObjMap|)(|r|)
  shows arr-Set α (umap-of 𝔉 c r u d)
proof(intro arr-SetI)
  interpret HomDom: category α 𝔄 by (rule assms(1))
  interpret HomCod: category α 𝔅 by (rule assms(2))
  note umap-of-components = umap-of-components[OF u]
  from u d have c: c ∈∘ 𝔅(|Obj|) and 𝔉d: (𝔉(|ObjMap|)(|d|)) ∈∘ 𝔅(|Obj|)
    by (auto intro: cat-cs-intros)
  show vfsequence (umap-of 𝔉 c r u d) unfolding umap-of-def by simp
  show vcard (umap-of 𝔉 c r u d) = 3_ℕ
    unfolding umap-of-def by (simp add: nat-omega-simps)
```

```
  from umap-of-ArrVal-vrange[OF u] show
    ℛ∘ (umap-of 𝔉 c r u d(|ArrVal|)) ⊆∘ umap-of 𝔉 c r u d(|ArrCod|)
    unfolding umap-of-components by simp
  from r d show umap-of 𝔉 c r u d(|ArrDom|) ∈∘ Vset α
    unfolding umap-of-components by (intro HomDom.cat-Hom-in-Vset)
  from c 𝔉d show umap-of 𝔉 c r u d(|ArrCod|) ∈∘ Vset α
    unfolding umap-of-components by (intro HomCod.cat-Hom-in-Vset)
qed (auto simp: umap-of-components[OF u])

lemma (in is-functor) cf-arr-Set-umap-fo:
  assumes category α 𝔄
    and category α 𝔅
    and r: r ∈∘ 𝔄(|Obj|)
    and d: d ∈∘ 𝔄(|Obj|)
    and u: u : 𝔉(|ObjMap|)(|r|) ↦𝔅 c
  shows arr-Set α (umap-fo 𝔉 c r u d)
proof–
  from assms(1) have 𝔄: category α (op-cat 𝔄)
    by (auto intro: cat-cs-intros)
  from assms(2) have 𝔅: category α (op-cat 𝔅)
    by (auto intro: cat-cs-intros)
  show ?thesis
    by
      (
        rule
          is-functor.cf-arr-Set-umap-of[
            OF is-functor-op, unfolded cat-op-simps, OF 𝔄 𝔅 r d u
            ]
      )
qed

lemma (in is-functor) cf-umap-of-is-arr:
  assumes category α 𝔄
    and category α 𝔅
    and r ∈∘ 𝔄(|Obj|)
    and d ∈∘ 𝔄(|Obj|)
    and u : c ↦𝔅 𝔉(|ObjMap|)(|r|)
  shows umap-of 𝔉 c r u d : Hom 𝔄 r d ↦cat-Set α Hom 𝔅 c (𝔉(|ObjMap|)(|d|))
proof(intro cat-Set-is-arrI)
  show arr-Set α (umap-of 𝔉 c r u d)
    by (rule cf-arr-Set-umap-of[OF assms])
qed (simp-all add: umap-of-components[OF assms(5)])

lemma (in is-functor) cf-umap-of-is-arr':
  assumes category α 𝔄
    and category α 𝔅
    and r ∈∘ 𝔄(|Obj|)
    and d ∈∘ 𝔄(|Obj|)
    and u : c ↦𝔅 𝔉(|ObjMap|)(|r|)
    and A = Hom 𝔄 r d
    and B = Hom 𝔅 c (𝔉(|ObjMap|)(|d|))
    and ℭ = cat-Set α
  shows umap-of 𝔉 c r u d : A ↦ℭ B
  using assms(1–5) unfolding assms(6–8) by (rule cf-umap-of-is-arr)

lemmas [cat-cs-intros] = is-functor.cf-umap-of-is-arr'

lemma (in is-functor) cf-umap-fo-is-arr:
```

assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $r \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $d \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
shows *umap-fo* $\mathfrak{F}$ c r u $d : Hom\ \mathfrak{A}\ d\ r \mapsto_{cat\text{-}Set\ \alpha} Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|d|\!))\ c$
proof(*intro cat-Set-is-arrI*)
show *arr-Set* α (*umap-fo* $\mathfrak{F}$ c r u d)
by (*rule cf-arr-Set-umap-fo*[*OF assms*])
qed (*simp-all add*: *umap-fo-components*[*OF assms*(*5*)])

lemma (**in** *is-functor*) *cf-umap-fo-is-arr'*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $r \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $d \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
and $A = Hom\ \mathfrak{A}\ d\ r$
and $B = Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|d|\!))\ c$
and $\mathfrak{C} = cat\text{-}Set\ \alpha$
shows *umap-fo* $\mathfrak{F}$ c r u $d : A \mapsto_{\mathfrak{C}} B$
using *assms*(*1–5*) **unfolding** *assms*(*6–8*) **by** (*rule cf-umap-fo-is-arr*)

lemmas [*cat-cs-intros*] = *is-functor.cf-umap-fo-is-arr'*

2.3 Universal arrow: definition and elementary properties

See Chapter III-1 in [9].

definition *universal-arrow-of* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
where *universal-arrow-of* $\mathfrak{F}$ c r $u \longleftrightarrow$
(
$r \in_\circ \mathfrak{F}(\!|HomDom|\!)(\!|Obj|\!) \wedge$
$u : c \mapsto_{\mathfrak{F}(\!|HomCod|\!)} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \wedge$
(
$\forall r'\ u'.$
$r' \in_\circ \mathfrak{F}(\!|HomDom|\!)(\!|Obj|\!) \longrightarrow$
$u' : c \mapsto_{\mathfrak{F}(\!|HomCod|\!)} \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!) \longrightarrow$
$(\exists!f'.\ f' : r \mapsto_{\mathfrak{F}(\!|HomDom|\!)} r' \wedge u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f'|\!))$
)
)

definition *universal-arrow-fo* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
where *universal-arrow-fo* $\mathfrak{F}$ c r $u \equiv$ *universal-arrow-of* (*op-cf* $\mathfrak{F}$) c r u

Rules.

mk-ide (**in** *is-functor*) **rf**
universal-arrow-of-def[**where** $\mathfrak{F}$=$\mathfrak{F}$, *unfolded cf-HomDom cf-HomCod*]
|*intro universal-arrow-ofI*|
|*dest universal-arrow-ofD*[*dest*]|
|*elim universal-arrow-ofE*[*elim*]|

lemma (**in** *is-functor*) *universal-arrow-foI*:
assumes $r \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
and $\bigwedge r'\ u'.\ [\![\ r' \in_\circ \mathfrak{A}(\!|Obj|\!);\ u' : \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!) \mapsto_{\mathfrak{B}} c\ ]\!] \Longrightarrow$
$\exists!f'.\ f' : r' \mapsto_{\mathfrak{A}} r \wedge u' = umap\text{-}fo\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f'|\!)$
shows *universal-arrow-fo* $\mathfrak{F}$ c r u

by
(
simp add:
is-functor.universal-arrow-ofI
[
OF is-functor-op,
folded universal-arrow-fo-def,
unfolded cat-op-simps,
OF assms
]
)

lemma (**in** *is-functor*) *universal-arrow-foD*[*dest*]:
assumes *universal-arrow-fo* $\mathfrak{F}$ *c r u*
shows $r \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
and $\bigwedge r' u'.\ [\![\ r' \in_{\circ} \mathfrak{A}(\!|Obj|\!);\ u' : \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!) \mapsto_{\mathfrak{B}} c\ ]\!] \Longrightarrow$
$\exists! f'.\ f' : r' \mapsto_{\mathfrak{A}} r \wedge u' = umap\text{-}fo\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f'|\!)$
by
(
auto simp:
is-functor.universal-arrow-ofD
[
OF is-functor-op,
folded universal-arrow-fo-def,
unfolded cat-op-simps,
OF assms
]
)

lemma (**in** *is-functor*) *universal-arrow-foE*[*elim*]:
assumes *universal-arrow-fo* $\mathfrak{F}$ *c r u*
obtains $r \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
and $\bigwedge r' u'.\ [\![\ r' \in_{\circ} \mathfrak{A}(\!|Obj|\!);\ u' : \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!) \mapsto_{\mathfrak{B}} c\ ]\!] \Longrightarrow$
$\exists! f'.\ f' : r' \mapsto_{\mathfrak{A}} r \wedge u' = umap\text{-}fo\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f'|\!)$
using *assms* **by** (*auto simp*: *universal-arrow-foD*)

Elementary properties.

lemma (**in** *is-functor*) *op-cf-universal-arrow-of*[*cat-op-simps*]:
universal-arrow-of (*op-cf* $\mathfrak{F}$) *c r u* $\longleftrightarrow$ *universal-arrow-fo* $\mathfrak{F}$ *c r u*
unfolding *universal-arrow-fo-def* **..**

lemma (**in** *is-functor*) *op-cf-universal-arrow-fo*[*cat-op-simps*]:
universal-arrow-fo (*op-cf* $\mathfrak{F}$) *c r u* $\longleftrightarrow$ *universal-arrow-of* $\mathfrak{F}$ *c r u*
unfolding *universal-arrow-fo-def cat-op-simps* **..**

lemmas (**in** *is-functor*) [*cat-op-simps*] =
is-functor.op-cf-universal-arrow-of
is-functor.op-cf-universal-arrow-fo

2.4 Uniqueness

The following properties are related to the uniqueness of the universal arrow. These properties can be inferred from the content of Chapter III-1 in [9].

lemma (**in** *is-functor*) *cf-universal-arrow-of-ex-is-iso-arr*:

— The proof is based on the ideas expressed in the proof of Theorem 5.2 in Chapter Introduction in [6].

assumes *universal-arrow-of* $\mathfrak{F}$ *c r u* **and** *universal-arrow-of* $\mathfrak{F}$ *c r′ u′*
obtains *f* **where** $f : r \mapsto_{iso\mathfrak{A}} r'$ **and** $u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f|\!)$
proof–

note *ua1* = *universal-arrow-ofD*[*OF assms*(*1*)]
note *ua2* = *universal-arrow-ofD*[*OF assms*(*2*)]

from *ua1*(*1*) **have** $\mathfrak{A}r$: $\mathfrak{A}(\!|CId|\!)(\!|r|\!) : r \mapsto_{\mathfrak{A}} r$ **by** (*auto intro*: *cat-cs-intros*)
from *ua1*(*1*) **have** $\mathfrak{F}(\!|ArrMap|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|r|\!)|\!) = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)|\!)$
by (*auto intro*: *cat-cs-intros*)
with *ua1*(*1,2*) **have** *u-def*: $u = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r(\!|ArrVal|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|r|\!)|\!)$
unfolding *umap-of-ArrVal-app*[*OF* $\mathfrak{A}r$ *ua1*(*2*)] **by** (*auto simp*: *cat-cs-simps*)

from *ua2*(*1*) **have** $\mathfrak{A}r'$: $\mathfrak{A}(\!|CId|\!)(\!|r'|\!) : r' \mapsto_{\mathfrak{A}} r'$ **by** (*auto intro*: *cat-cs-intros*)
from *ua2*(*1*) **have** $\mathfrak{F}(\!|ArrMap|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|r'|\!)|\!) = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!)|\!)$
by (*auto intro*: *cat-cs-intros*)
with *ua2*(*1,2*) **have** *u′-def*: $u' = umap\text{-}of\ \mathfrak{F}\ c\ r'\ u'\ r'(\!|ArrVal|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|r'|\!)|\!)$
unfolding *umap-of-ArrVal-app*[*OF* $\mathfrak{A}r'$ *ua2*(*2*)] **by** (*auto simp*: *cat-cs-simps*)

from $\mathfrak{A}r$ *u-def universal-arrow-ofD*(*3*)[*OF assms*(*1*) *ua1*(*1,2*)] **have** *eq-CId-rI*:
$[\![\ f' : r \mapsto_{\mathfrak{A}} r;\ u = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r(\!|ArrVal|\!)(\!|f'|\!)\ ]\!] \Longrightarrow f' = \mathfrak{A}(\!|CId|\!)(\!|r|\!)$
for *f′*
by *blast*
from $\mathfrak{A}r'$ *u′-def universal-arrow-ofD*(*3*)[*OF assms*(*2*) *ua2*(*1,2*)] **have** *eq-CId-r′I*:
$[\![\ f' : r' \mapsto_{\mathfrak{A}} r';\ u' = umap\text{-}of\ \mathfrak{F}\ c\ r'\ u'\ r'(\!|ArrVal|\!)(\!|f'|\!)\ ]\!] \Longrightarrow$
$f' = \mathfrak{A}(\!|CId|\!)(\!|r'|\!)$
for *f′*
by *blast*

from *ua1*(*3*)[*OF ua2*(*1,2*)] **obtain** *f*
where *f*: $f : r \mapsto_{\mathfrak{A}} r'$
and *u′-def*: $u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f|\!)$
and $g : r \mapsto_{\mathfrak{A}} r' \Longrightarrow u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|g|\!) \Longrightarrow f = g$
for *g*
by *metis*
from *ua2*(*3*)[*OF ua1*(*1,2*)] **obtain** *f′*
where *f′*: $f' : r' \mapsto_{\mathfrak{A}} r$
and *u-def*: $u = umap\text{-}of\ \mathfrak{F}\ c\ r'\ u'\ r(\!|ArrVal|\!)(\!|f'|\!)$
and $g : r' \mapsto_{\mathfrak{A}} r \Longrightarrow u = umap\text{-}of\ \mathfrak{F}\ c\ r'\ u'\ r(\!|ArrVal|\!)(\!|g|\!) \Longrightarrow f' = g$
for *g*
by *metis*

have $f : r \mapsto_{iso\mathfrak{A}} r'$
proof(*intro is-iso-arrI is-inverseI*)
show *f*: $f : r \mapsto_{\mathfrak{A}} r'$ **by** (*rule f*)
show *f′*: $f' : r' \mapsto_{\mathfrak{A}} r$ **by** (*rule f′*)
show $f : r \mapsto_{\mathfrak{A}} r'$ **by** (*rule f*)
from *f′* **have** $\mathfrak{F}f'$: $\mathfrak{F}(\!|ArrMap|\!)(\!|f'|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
by (*auto intro*: *cat-cs-intros*)
from *f* **have** $\mathfrak{F}f$: $\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!)$
by (*auto intro*: *cat-cs-intros*)
note *u′-def′* = *u′-def*[*symmetric*, *unfolded umap-of-ArrVal-app*[*OF f ua1*(*2*)]]
and *u-def′* = *u-def*[*symmetric*, *unfolded umap-of-ArrVal-app*[*OF f′ ua2*(*2*)]]
show $f' \circ_{A\mathfrak{A}} f = \mathfrak{A}(\!|CId|\!)(\!|r|\!)$
proof(*rule eq-CId-rI*)
from *f f′* **show** *f′f*: $f' \circ_{A\mathfrak{A}} f : r \mapsto_{\mathfrak{A}} r$

```
      by (auto intro: cat-cs-intros)
    from ua1(2) ℱf' ℱf show u = umap-of ℱ c r u r(|ArrVal|)(|f' ∘_A𝔄 f|)
      unfolding umap-of-ArrVal-app[OF f'f ua1(2)] cf-ArrMap-Comp[OF f' f]
      by (simp add: HomCod.cat-Comp-assoc u'-def' u-def')
  qed
  show f ∘_A𝔄 f' = 𝔄(|CId|)(|r'|)
  proof(rule eq-CId-r'I)
    from f f' show ff': f ∘_A𝔄 f' : r' ↦_𝔄 r'
      by (auto intro: cat-cs-intros)
    from ua2(2) ℱf' ℱf show u' = umap-of ℱ c r' u' r'(|ArrVal|)(|f ∘_A𝔄 f'|)
      unfolding umap-of-ArrVal-app[OF ff' ua2(2)] cf-ArrMap-Comp[OF f f']
      by (simp add: HomCod.cat-Comp-assoc u'-def' u-def')
  qed
qed

  with u'-def that show ?thesis by auto

qed

lemma (in is-functor) cf-universal-arrow-fo-ex-is-iso-arr:
  assumes universal-arrow-fo ℱ c r u
    and universal-arrow-fo ℱ c r' u'
  obtains f where f : r' ↦_iso𝔄 r and u' = umap-fo ℱ c r u r'(|ArrVal|)(|f|)
  by
    (
      elim
        is-functor.cf-universal-arrow-of-ex-is-iso-arr[
          OF is-functor-op, unfolded cat-op-simps, OF assms
          ]
    )

lemma (in is-functor) cf-universal-arrow-of-unique:
  assumes universal-arrow-of ℱ c r u
    and universal-arrow-of ℱ c r' u'
  shows ∃!f'. f' : r ↦_𝔄 r' ∧ u' = umap-of ℱ c r u r'(|ArrVal|)(|f'|)
proof–
  note ua1 = universal-arrow-ofD[OF assms(1)]
  note ua2 = universal-arrow-ofD[OF assms(2)]
  from ua1(3)[OF ua2(1,2)] show ?thesis .
qed

lemma (in is-functor) cf-universal-arrow-fo-unique:
  assumes universal-arrow-fo ℱ c r u
    and universal-arrow-fo ℱ c r' u'
  shows ∃!f'. f' : r' ↦_𝔄 r ∧ u' = umap-fo ℱ c r u r'(|ArrVal|)(|f'|)
proof–
  note ua1 = universal-arrow-foD[OF assms(1)]
  note ua2 = universal-arrow-foD[OF assms(2)]
  from ua1(3)[OF ua2(1,2)] show ?thesis .
qed

lemma (in is-functor) cf-universal-arrow-of-is-iso-arr:
  assumes universal-arrow-of ℱ c r u
    and universal-arrow-of ℱ c r' u'
    and f : r ↦_𝔄 r'
    and u' = umap-of ℱ c r u r'(|ArrVal|)(|f|)
  shows f : r ↦_iso𝔄 r'
proof–
```

```
  from assms(3,4) cf-universal-arrow-of-unique[OF assms(1,2)] have eq:
    g : r ↦_𝔄 r' ⟹ u' = umap-of 𝔉 c r u r'(|ArrVal|)(|g|) ⟹ f = g for g
    by blast
  from assms(1,2) obtain f'
    where iso-f': f' : r ↦_iso𝔄 r'
      and u'-def: u' = umap-of 𝔉 c r u r'(|ArrVal|)(|f'|)
    by (auto elim: cf-universal-arrow-of-ex-is-iso-arr)
  then have f': f' : r ↦_𝔄 r' by auto
  from iso-f' show ?thesis unfolding eq[OF f' u'-def, symmetric].
qed

lemma (in is-functor) cf-universal-arrow-fo-is-iso-arr:
  assumes universal-arrow-fo 𝔉 c r u
    and universal-arrow-fo 𝔉 c r' u'
    and f : r' ↦_𝔄 r
    and u' = umap-fo 𝔉 c r u r'(|ArrVal|)(|f|)
  shows f : r' ↦_iso𝔄 r
  by
    (
      rule
        is-functor.cf-universal-arrow-of-is-iso-arr[
          OF is-functor-op, unfolded cat-op-simps, OF assms
          ]
    )

lemma (in is-functor) universal-arrow-of-if-universal-arrow-of:
  assumes universal-arrow-of 𝔉 c r u
    and f : r ↦_iso𝔄 r'
    and u' = umap-of 𝔉 c r u r'(|ArrVal|)(|f|)
  shows universal-arrow-of 𝔉 c r' u'
proof(intro universal-arrow-ofI assms(2))

  note ua = universal-arrow-ofD[OF assms(1)]
  note f = is-iso-arrD(1)[OF assms(2)]
  from assms(3) ua(1,2) f have u'-def: u' = 𝔉(|ArrMap|)(|f|) ∘_A𝔅 u
    by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from ua(2) f show u': u' : c ↦_𝔅 𝔉(|ObjMap|)(|r'|)
    unfolding u'-def by (cs-concl cs-intro: cat-cs-intros)

  from f(1) show r' ∈_∘ 𝔄(|Obj|) by auto

  fix r'' u'' assume prems: r'' ∈_∘ 𝔄(|Obj|) u'' : c ↦_𝔅 𝔉(|ObjMap|)(|r''|)

  from ua(3)[OF prems] obtain f'
    where f': f' : r ↦_𝔄 r''
      and u''-def: u'' = umap-of 𝔉 c r u r''(|ArrVal|)(|f'|)
      and f'-unique: ⋀f''.
        [[ f'' : r ↦_𝔄 r''; u'' = umap-of 𝔉 c r u r''(|ArrVal|)(|f''|) ]] ⟹
          f'' = f'
    by metis

  from u''-def f' ua(2) have [cat-cs-simps]: 𝔉(|ArrMap|)(|f'|) ∘_A𝔅 u = u''
    by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros) simp

  show ∃!f'. f' : r' ↦_𝔄 r'' ∧ u'' = umap-of 𝔉 c r' u' r''(|ArrVal|)(|f'|)
  proof(intro ex1I conjI; (elim conjE)?)
    from f' assms(2) f show f' ∘_A𝔄 f⁻¹_C𝔄 : r' ↦_𝔄 r''
      by (cs-concl cs-intro: cat-cs-intros cat-arrow-cs-intros)
```

```
    have
      𝔉(⌊ArrMap⌋)(⌊f'⌋) ∘_{A𝔅} (𝔉(⌊ArrMap⌋)(⌊f^-1_{C𝔄}⌋) ∘_{A𝔅} u') =
        𝔉(⌊ArrMap⌋)(⌊f'⌋) ∘_{A𝔅} (𝔉(⌊ArrMap⌋)(⌊f^-1_{C𝔄}⌋) ∘_{A𝔅} (𝔉(⌊ArrMap⌋)(⌊f⌋) ∘_{A𝔅} u))
      unfolding u'-def ..
    also from f' assms(2) u' f ua(2) have
      ... = 𝔉(⌊ArrMap⌋)(⌊f'⌋) ∘_{A𝔅} (𝔉(⌊ArrMap⌋)(⌊f^-1_{C𝔄} ∘_{A𝔄} f⌋)) ∘_{A𝔅} u
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cs-intro: cat-arrow-cs-intros cat-cs-intros
        )
    also from f' assms(2) f ua(2) have ... = u''
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    finally have [cat-cs-simps]:
      𝔉(⌊ArrMap⌋)(⌊f'⌋) ∘_{A𝔅} (𝔉(⌊ArrMap⌋)(⌊f^-1_{C𝔄}⌋) ∘_{A𝔅} u') = u''.
    from f' assms(2) u' f show
      u'' = umap-of 𝔉 c r' u' r''(⌊ArrVal⌋)(⌊f' ∘_{A𝔄} f^-1_{C𝔄}⌋)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-arrow-cs-intros
        )
    fix g assume prems':
      g : r' ↦_𝔄 r'' u'' = umap-of 𝔉 c r' u' r''(⌊ArrVal⌋)(⌊g⌋)
    from prems'(1) f have gf: g ∘_{A𝔄} f : r ↦_𝔄 r''
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from prems'(2,1) assms(2) u' have u'' = 𝔉(⌊ArrMap⌋)(⌊g⌋) ∘_{A𝔅} u'
      by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    also from prems'(1) f ua(2) have
      ... = 𝔉(⌊ArrMap⌋)(⌊g⌋) ∘_{A𝔅} 𝔉(⌊ArrMap⌋)(⌊f⌋) ∘_{A𝔅} u
      by (cs-concl cs-simp: cat-cs-simps u'-def u''-def cs-intro: cat-cs-intros)
    also from prems'(1) f ua(2) have
      ... = umap-of 𝔉 c r u r''(⌊ArrVal⌋)(⌊g ∘_{A𝔄} f⌋)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    finally have u'' = umap-of 𝔉 c r u r''(⌊ArrVal⌋)(⌊g ∘_{A𝔄} f⌋).
    from f'-unique[OF gf this] have g ∘_{A𝔄} f = f'.
    then have (g ∘_{A𝔄} f) ∘_{A𝔄} f^-1_{C𝔄} = f' ∘_{A𝔄} f^-1_{C𝔄} by simp
    from this assms(2) prems'(1) u' f ua(2) show g = f' ∘_{A𝔄} f^-1_{C𝔄}
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cs-intro: cat-arrow-cs-intros cat-cs-intros
        )
  qed

qed

lemma (in is-functor) universal-arrow-fo-if-universal-arrow-fo:
  assumes universal-arrow-fo 𝔉 c r u
    and f : r' ↦_{iso𝔄} r
    and u' = umap-fo 𝔉 c r u r'(⌊ArrVal⌋)(⌊f⌋)
  shows universal-arrow-fo 𝔉 c r' u'
  by
    (
      rule is-functor.universal-arrow-of-if-universal-arrow-of[
        OF is-functor-op, unfolded cat-op-simps, OF assms
        ]
    )
```

2.5 Universal natural transformation

2.5.1 Definition and elementary properties

The concept of the universal natural transformation is introduced for the statement of the elements of a variant of Proposition 1 in Chapter III-2 in [9].

definition *ntcf-ua-of* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-ua-of* $\alpha\ \mathfrak{F}\ c\ r\ u =$
    [
      $(\lambda d \in_\circ \mathfrak{F}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ d)$,
      $Hom_{O.C\alpha}\mathfrak{F}(\lfloor HomDom \rfloor)(r,-)$,
      $Hom_{O.C\alpha}\mathfrak{F}(\lfloor HomCod \rfloor)(c,-) \circ_{CF} \mathfrak{F}$,
      $\mathfrak{F}(\lfloor HomDom \rfloor)$,
      $cat\text{-}Set\ \alpha$
    ]$_\circ$

definition *ntcf-ua-fo* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-ua-fo* $\alpha\ \mathfrak{F}\ c\ r\ u = ntcf\text{-}ua\text{-}of\ \alpha\ (op\text{-}cf\ \mathfrak{F})\ c\ r\ u$

Components.

lemma *ntcf-ua-of-components*:
  shows $ntcf\text{-}ua\text{-}of\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTMap \rfloor) = (\lambda d \in_\circ \mathfrak{F}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ d)$
    and $ntcf\text{-}ua\text{-}of\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDom \rfloor) = Hom_{O.C\alpha}\mathfrak{F}(\lfloor HomDom \rfloor)(r,-)$
    and $ntcf\text{-}ua\text{-}of\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTCod \rfloor) = Hom_{O.C\alpha}\mathfrak{F}(\lfloor HomCod \rfloor)(c,-) \circ_{CF} \mathfrak{F}$
    and $ntcf\text{-}ua\text{-}of\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDGDom \rfloor) = \mathfrak{F}(\lfloor HomDom \rfloor)$
    and $ntcf\text{-}ua\text{-}of\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDGCod \rfloor) = cat\text{-}Set\ \alpha$
  unfolding *ntcf-ua-of-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *ntcf-ua-fo-components*:
  shows $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTMap \rfloor) = (\lambda d \in_\circ \mathfrak{F}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ umap\text{-}fo\ \mathfrak{F}\ c\ r\ u\ d)$
    and $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDom \rfloor) = Hom_{O.C\alpha}op\text{-}cat\ (\mathfrak{F}(\lfloor HomDom \rfloor))(r,-)$
    and $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTCod \rfloor) =$
      $Hom_{O.C\alpha}op\text{-}cat\ (\mathfrak{F}(\lfloor HomCod \rfloor))(c,-) \circ_{CF} op\text{-}cf\ \mathfrak{F}$
    and $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDGDom \rfloor) = op\text{-}cat\ (\mathfrak{F}(\lfloor HomDom \rfloor))$
    and $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDGCod \rfloor) = cat\text{-}Set\ \alpha$
  unfolding *ntcf-ua-fo-def ntcf-ua-of-components umap-fo-def cat-op-simps*
  by *simp-all*

context *is-functor*
begin

lemmas *ntcf-ua-of-components'* =
  ntcf-ua-of-components[**where** α=α **and** $\mathfrak{F}$=$\mathfrak{F}$, *unfolded cat-cs-simps*]

lemmas [*cat-cs-simps*] = *ntcf-ua-of-components'*(*2–5*)

lemma *ntcf-ua-fo-components'*:
  assumes $c \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$ **and** $r \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
  shows $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTMap \rfloor) = (\lambda d \in_\circ \mathfrak{A}(\lfloor Obj \rfloor).\ umap\text{-}fo\ \mathfrak{F}\ c\ r\ u\ d)$
    and [*cat-cs-simps*]:
      $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDom \rfloor) = Hom_{O.C\alpha}\mathfrak{A}(-,r)$
    and [*cat-cs-simps*]:
      $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTCod \rfloor) = Hom_{O.C\alpha}\mathfrak{B}(-,c) \circ_{CF} op\text{-}cf\ \mathfrak{F}$
    and [*cat-cs-simps*]: $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDGDom \rfloor) = op\text{-}cat\ \mathfrak{A}$
    and [*cat-cs-simps*]: $ntcf\text{-}ua\text{-}fo\ \alpha\ \mathfrak{F}\ c\ r\ u(\lfloor NTDGCod \rfloor) = cat\text{-}Set\ \alpha$
  unfolding
    ntcf-ua-fo-components cat-cs-simps
    HomDom.cat-op-cat-cf-Hom-snd[*OF assms*(*2*)]

HomCod.cat-op-cat-cf-Hom-snd[*OF assms*(*1*)]
by *simp-all*

end

lemmas [*cat-cs-simps*] =
is-functor.ntcf-ua-of-components'(*2–5*)
is-functor.ntcf-ua-fo-components'(*2–5*)

2.5.2 Natural transformation map

mk-VLambda (**in** *is-functor*)
ntcf-ua-of-components(*1*)[**where** α=α **and** $\mathfrak{F}$=$\mathfrak{F}$, *unfolded cf-HomDom*]
|*vsv ntcf-ua-of-NTMap-vsv*|
|*vdomain ntcf-ua-of-NTMap-vdomain*|
|*app ntcf-ua-of-NTMap-app*|

context *is-functor*
begin

context
fixes *c r*
assumes *r*: $r \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ **and** *c*: $c \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
begin

mk-VLambda *ntcf-ua-fo-components'*(*1*)[*OF c r*]
|*vsv ntcf-ua-fo-NTMap-vsv*|
|*vdomain ntcf-ua-fo-NTMap-vdomain*|
|*app ntcf-ua-fo-NTMap-app*|

end

end

lemmas [*cat-cs-intros*] =
is-functor.ntcf-ua-fo-NTMap-vsv
is-functor.ntcf-ua-of-NTMap-vsv

lemmas [*cat-cs-simps*] =
is-functor.ntcf-ua-fo-NTMap-vdomain
is-functor.ntcf-ua-fo-NTMap-app
is-functor.ntcf-ua-of-NTMap-vdomain
is-functor.ntcf-ua-of-NTMap-app

lemma (**in** *is-functor*) *ntcf-ua-of-NTMap-vrange*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $r \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
shows $\mathcal{R}_{\circ}$ (*ntcf-ua-of* α $\mathfrak{F}$ *c r* $u(\!|NTMap|\!)$) $\subseteq_{\circ}$ *cat-Set* $\alpha(\!|Arr|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold ntcf-ua-of-NTMap-vdomain*)
show *vsv* (*ntcf-ua-of* α $\mathfrak{F}$ *c r* $u(\!|NTMap|\!)$) **by** (*rule ntcf-ua-of-NTMap-vsv*)
fix *d* **assume** *prems*: $d \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
with *category-cat-Set is-functor-axioms assms* **show**
ntcf-ua-of α $\mathfrak{F}$ *c r* $u(\!|NTMap|\!)(\!|d|\!) \in_{\circ}$ *cat-Set* $\alpha(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

2.5.3 Commutativity of the universal maps and *hom*-functions

lemma (**in** *is-functor*) *cf-umap-of-cf-hom-commute*:
  assumes *category* α $\mathfrak{A}$
    and *category* α $\mathfrak{B}$
    and $c \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
    and $r \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
    and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
    and $f : a \mapsto_{\mathfrak{A}} b$
  shows
    umap-of $\mathfrak{F}$ c r u b $\circ_{A\,cat\text{-}Set\ \alpha}$ *cf-hom* $\mathfrak{A}$ $[\mathfrak{A}(\!|CId|\!)(\!|r|\!),\ f]_{\circ}$ =
      cf-hom $\mathfrak{B}$ $[\mathfrak{B}(\!|CId|\!)(\!|c|\!),\ \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)]_{\circ}$ $\circ_{A\,cat\text{-}Set\ \alpha}$ *umap-of* $\mathfrak{F}$ c r u a
  (**is** ‹*?uof-b* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?rf* = *?cf* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?uof-a*›)
proof–

  from *is-functor-axioms category-cat-Set assms*(*1*,*2*,*4*–*6*) **have** *b-rf*:
    ?uof-b $\circ_{A\,cat\text{-}Set\ \alpha}$ *?rf* : *Hom* $\mathfrak{A}$ r a $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{B}$ c $(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
    by
      (
        cs-concl **cs-shallow**
          cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
  from *is-functor-axioms category-cat-Set assms*(*1*,*2*,*4*–*6*) **have** *cf-a*:
    ?cf $\circ_{A\,cat\text{-}Set\ \alpha}$ *?uof-a* : *Hom* $\mathfrak{A}$ r a $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{B}$ c $(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
    by (*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros cat-prod-cs-intros*)

  show *?thesis*
  proof(*rule arr-Set-eqI*[*of* α])
    from *b-rf* **show** *arr-Set-b-rf*: *arr-Set* α (*?uof-b* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?rf*)
      by (*auto dest*: *cat-Set-is-arrD*(*1*))
    from *b-rf* **have** *dom-lhs*:
      $\mathcal{D}_{\circ}$ ((*?uof-b* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?rf*)(|*ArrVal*|)) = *Hom* $\mathfrak{A}$ r a
      by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)+
    from *cf-a* **show** *arr-Set-cf-a*: *arr-Set* α (*?cf* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?uof-a*)
      by (*auto dest*: *cat-Set-is-arrD*(*1*))
    from *cf-a* **have** *dom-rhs*:
      $\mathcal{D}_{\circ}$ ((*?cf* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?uof-a*)(|*ArrVal*|)) = *Hom* $\mathfrak{A}$ r a
      by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
    show (*?uof-b* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?rf*)(|*ArrVal*|) = (*?cf* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?uof-a*)(|*ArrVal*|)
    proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs in-Hom-iff*)
      fix q **assume** $q : r \mapsto_{\mathfrak{A}} a$
      with *is-functor-axioms category-cat-Set assms* **show**
        (*?uof-b* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?rf*)(|*ArrVal*|)(|q|) =
          (*?cf* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?uof-a*)(|*ArrVal*|)(|q|)
        by
          (
            cs-concl **cs-shallow**
              cs-simp: *cat-cs-simps cat-op-simps*
              cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
    qed (*use arr-Set-b-rf arr-Set-cf-a* **in** *auto*)

  qed (*use b-rf cf-a* **in** ‹*cs-concl cs-shallow cs-simp*: *cat-cs-simps*›)+

qed

lemma *cf-umap-of-cf-hom-unit-commute*:
  assumes *category* α $\mathfrak{C}$

```
    and category α 𝔇
    and 𝔉 : ℭ ↦↦_{Cα} 𝔇
    and 𝔊 : 𝔇 ↦↦_{Cα} ℭ
    and η : cf-id ℭ ↦_{CF} 𝔊 ∘_{CF} 𝔉 : ℭ ↦↦_{Cα} ℭ
    and g : c' ↦_ℭ c
    and f : d ↦_𝔇 d'
  shows
    umap-of 𝔊 c' (𝔉(|ObjMap|)(|c'|)) (η(|NTMap|)(|c'|)) d' ∘_{A cat-Set α}
      cf-hom 𝔇 [𝔉(|ArrMap|)(|g|), f]_∘ =
        cf-hom ℭ [g, 𝔊(|ArrMap|)(|f|)]_∘ ∘_{A cat-Set α}
          umap-of 𝔊 c (𝔉(|ObjMap|)(|c|)) (η(|NTMap|)(|c|)) d
  (is ‹?uof-c'd' ∘_{A cat-Set α} ?𝔉gf = ?g𝔊f ∘_{A cat-Set α} ?uof-cd›)
proof–

  interpret η: is-ntcf α ℭ ℭ ‹cf-id ℭ› ‹𝔊 ∘_{CF} 𝔉› η by (rule assms(5))

  from assms have c'd'-𝔉gf: ?uof-c'd' ∘_{A cat-Set α} ?𝔉gf :
    Hom 𝔇 (𝔉(|ObjMap|)(|c|)) d ↦_{cat-Set α} Hom ℭ c' (𝔊(|ObjMap|)(|d'|))
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  then have dom-lhs:
    𝒟∘ ((?uof-c'd' ∘_{A cat-Set α} ?𝔉gf)(|ArrVal|)) = Hom 𝔇 (𝔉(|ObjMap|)(|c|)) d
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms have g𝔊f-cd: ?g𝔊f ∘_{A cat-Set α} ?uof-cd :
    Hom 𝔇 (𝔉(|ObjMap|)(|c|)) d ↦_{cat-Set α} Hom ℭ c' (𝔊(|ObjMap|)(|d'|))
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  then have dom-rhs:
    𝒟∘ ((?g𝔊f ∘_{A cat-Set α} ?uof-cd)(|ArrVal|)) = Hom 𝔇 (𝔉(|ObjMap|)(|c|)) d
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)

  show ?thesis
  proof(rule arr-Set-eqI[of α])
    from c'd'-𝔉gf show arr-Set-c'd'-𝔉gf:
      arr-Set α (?uof-c'd' ∘_{A cat-Set α} ?𝔉gf)
      by (auto dest: cat-Set-is-arrD(1))
    from g𝔊f-cd show arr-Set-g𝔊f-cd:
      arr-Set α (?g𝔊f ∘_{A cat-Set α} ?uof-cd)
      by (auto dest: cat-Set-is-arrD(1))
    show
      (?uof-c'd' ∘_{A cat-Set α} ?𝔉gf)(|ArrVal|) =
        (?g𝔊f ∘_{A cat-Set α} ?uof-cd)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix h assume prems: h : 𝔉(|ObjMap|)(|c|) ↦_𝔇 d
      from η.ntcf-Comp-commute[OF assms(6)] assms have [cat-cs-simps]:
        η(|NTMap|)(|c|) ∘_{Aℭ} g = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|g|)|) ∘_{Aℭ} η(|NTMap|)(|c'|)
        by
          (
            cs-prems cs-shallow
              cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
```

```
      )
    from assms prems show
      (?uof-c′d′ ∘_A cat-Set α ?𝔉gf)(|ArrVal|)(|h|) =
        (?g𝔊f ∘_A cat-Set α ?uof-cd)(|ArrVal|)(|h|)
      by
        (
          cs-concl
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
            cs-simp: cat-cs-simps
        )
  qed (use arr-Set-c′d′-𝔉gf arr-Set-g𝔊f-cd in auto)

 qed (use c′d′-𝔉gf g𝔊f-cd in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed
```

2.5.4 Universal natural transformation is a natural transformation

```
lemma (in is-functor) cf-ntcf-ua-of-is-ntcf:
  assumes r ∈_∘ 𝔄(|Obj|)
    and u : c ↦_𝔅 𝔉(|ObjMap|)(|r|)
  shows ntcf-ua-of α 𝔉 c r u :
    Hom_{O.Cα}𝔄(r,−) ↦_CF Hom_{O.Cα}𝔅(c,−) ∘_CF 𝔉 : 𝔄 ↦↦_Cα cat-Set α
proof(intro is-ntcfI′)
  let ?ua = ‹ntcf-ua-of α 𝔉 c r u›
  show vfsequence (ntcf-ua-of α 𝔉 c r u) unfolding ntcf-ua-of-def by simp
  show vcard ?ua = 5_ℕ unfolding ntcf-ua-of-def by (simp add: nat-omega-simps)
  from assms(1) show Hom_{O.Cα}𝔄(r,−) : 𝔄 ↦↦_Cα cat-Set α
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from is-functor-axioms assms(2) show
    Hom_{O.Cα}𝔅(c,−) ∘_CF 𝔉 : 𝔄 ↦↦_Cα cat-Set α
    by (cs-concl cs-intro: cat-cs-intros)
  from is-functor-axioms assms show 𝒟_∘ (?ua(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?ua(|NTMap|)(|a|) :
    Hom_{O.Cα}𝔄(r,−)(|ObjMap|)(|a|) ↦_{cat-Set α} (Hom_{O.Cα}𝔅(c,−) ∘_CF 𝔉)(|ObjMap|)(|a|)
    if a ∈_∘ 𝔄(|Obj|) for a
    using is-functor-axioms assms that
    by (cs-concl cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros)
  show ?ua(|NTMap|)(|b|) ∘_{A cat-Set α} Hom_{O.Cα}𝔄(r,−)(|ArrMap|)(|f|) =
    (Hom_{O.Cα}𝔅(c,−) ∘_CF 𝔉)(|ArrMap|)(|f|) ∘_{A cat-Set α} ?ua(|NTMap|)(|a|)
    if f : a ↦_𝔄 b for a b f
    using is-functor-axioms assms that
    by
      (
        cs-concl
          cs-simp: cf-umap-of-cf-hom-commute cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
qed (auto simp: ntcf-ua-of-components cat-cs-simps)

lemma (in is-functor) cf-ntcf-ua-fo-is-ntcf:
  assumes r ∈_∘ 𝔄(|Obj|) and u : 𝔉(|ObjMap|)(|r|) ↦_𝔅 c
  shows ntcf-ua-fo α 𝔉 c r u :
    Hom_{O.Cα}𝔄(−,r) ↦_CF Hom_{O.Cα}𝔅(−,c) ∘_CF op-cf 𝔉 :
    op-cat 𝔄 ↦↦_Cα cat-Set α
proof–
  from assms(2) have c: c ∈_∘ 𝔅(|Obj|) by auto
```

```
  show ?thesis
    by
      (
        rule is-functor.cf-ntcf-ua-of-is-ntcf
          [
            OF is-functor-op,
            unfolded cat-op-simps,
            OF assms(1,2),
            unfolded
              HomDom.cat-op-cat-cf-Hom-snd[OF assms(1)]
              HomCod.cat-op-cat-cf-Hom-snd[OF c]
              ntcf-ua-fo-def[symmetric]
          ]
      )
qed
```

2.5.5 Universal natural transformation and universal arrow

The lemmas in this subsection correspond to variants of elements of Proposition 1 in Chapter III-2 in [9].

```
lemma (in is-functor) cf-ntcf-ua-of-is-iso-ntcf:
  assumes universal-arrow-of 𝔉 c r u
  shows ntcf-ua-of α 𝔉 c r u :
    Hom_{O.Cα}𝔄(r,−) ↦_{CF.iso} Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉 : 𝔄 ↦↦_{Cα} cat-Set α
proof–

  have r: r ∈ₒ 𝔄(|Obj|)
    and u: u : c ↦_𝔅 𝔉(|ObjMap|)(|r|)
    and bij: ⋀r' u'.
      [[
        r' ∈ₒ 𝔄(|Obj|);
        u' : c ↦_𝔅 𝔉(|ObjMap|)(|r'|)
      ]] ⟹ ∃!f'. f' : r ↦_𝔄 r' ∧ u' = umap-of 𝔉 c r u r'(|ArrVal|)(|f'|)
    by (auto intro!: universal-arrow-ofD[OF assms(1)])

  show ?thesis
  proof(intro is-iso-ntcfI)
    show ntcf-ua-of α 𝔉 c r u :
      Hom_{O.Cα}𝔄(r,−) ↦_{CF} Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉 : 𝔄 ↦↦_{Cα} cat-Set α
      by (rule cf-ntcf-ua-of-is-ntcf[OF r u])
    fix a assume prems: a ∈ₒ 𝔄(|Obj|)
    from is-functor-axioms prems r u have [simp]:
      umap-of 𝔉 c r u a : Hom 𝔄 r a ↦_{cat-Set α} Hom 𝔅 c (𝔉(|ObjMap|)(|a|))
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    then have dom: 𝒟ₒ (umap-of 𝔉 c r u a(|ArrVal|)) = Hom 𝔄 r a
      by (cs-concl cs-simp: cat-cs-simps)
    have umap-of 𝔉 c r u a : Hom 𝔄 r a ↦_{iso cat-Set α} Hom 𝔅 c (𝔉(|ObjMap|)(|a|))
    proof(intro cat-Set-is-iso-arrI, unfold dom)

      show umof-a: v11 (umap-of 𝔉 c r u a(|ArrVal|))
      proof(intro vsv.vsv-valeq-v11I, unfold dom in-Hom-iff)
        fix g f assume prems':
          g : r ↦_𝔄 a
          f : r ↦_𝔄 a
          umap-of 𝔉 c r u a(|ArrVal|)(|g|) = umap-of 𝔉 c r u a(|ArrVal|)(|f|)
        from is-functor-axioms r u prems'(1) have 𝔉g:
          𝔉(|ArrMap|)(|g|) ∘_{A𝔅} u : c ↦_𝔅 𝔉(|ObjMap|)(|a|)
```

```
        by (cs-concl cs-shallow cs-intro: cat-cs-intros)
      from bij[OF prems 𝔉g] have unique:
        ⟦
          f′ : r ↦₍𝔄₎ a;
          𝔉(|ArrMap|)(|g|) ∘₍A𝔅₎ u = umap-of 𝔉 c r u a(|ArrVal|)(|f′|)
        ⟧ ⟹ g = f′
        for f′ by (metis prems′(1) u umap-of-ArrVal-app)
      from is-functor-axioms prems′(1,2) u have 𝔉g-u:
        𝔉(|ArrMap|)(|g|) ∘₍A𝔅₎ u = umap-of 𝔉 c r u a(|ArrVal|)(|f|)
        by (cs-concl cs-simp: prems′(3)[symmetric] cat-cs-simps)
      show g = f by (rule unique[OF prems′(2) 𝔉g-u])
    qed (auto simp: cat-cs-simps cat-cs-intros)

    interpret umof-a: v11 ‹umap-of 𝔉 c r u a(|ArrVal|)› by (rule umof-a)

    show ℛ∘ (umap-of 𝔉 c r u a(|ArrVal|)) = Hom 𝔅 c (𝔉(|ObjMap|)(|a|))
    proof(intro vsubset-antisym)
      from u show ℛ∘ (umap-of 𝔉 c r u a(|ArrVal|)) ⊆∘ Hom 𝔅 c (𝔉(|ObjMap|)(|a|))
        by (rule umap-of-ArrVal-vrange)
      show Hom 𝔅 c (𝔉(|ObjMap|)(|a|)) ⊆∘ ℛ∘ (umap-of 𝔉 c r u a(|ArrVal|))
      proof(rule vsubsetI, unfold in-Hom-iff )
        fix f assume prems′: f : c ↦₍𝔅₎ 𝔉(|ObjMap|)(|a|)
        from bij[OF prems prems′] obtain f′
          where f′: f′ : r ↦₍𝔄₎ a
            and f-def: f = umap-of 𝔉 c r u a(|ArrVal|)(|f′|)
          by auto
        from is-functor-axioms prems prems′ u f′ have
          f′ ∈∘ 𝒟∘ (umap-of 𝔉 c r u a(|ArrVal|))
          by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
        from this show f ∈∘ ℛ∘ (umap-of 𝔉 c r u a(|ArrVal|))
          unfolding f-def by (rule umof-a.vsv-vimageI2)
      qed

    qed

  qed simp-all

  from is-functor-axioms prems r u this show
    ntcf-ua-of α 𝔉 c r u(|NTMap|)(|a|) :
      Hom_{O.Cα}𝔄(r,−)(|ObjMap|)(|a|) ↦_{iso cat-Set α}
      (Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉)(|ObjMap|)(|a|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
qed

qed

lemmas [cat-cs-intros] = is-functor.cf-ntcf-ua-of-is-iso-ntcf

lemma (in is-functor) cf-ntcf-ua-fo-is-iso-ntcf:
  assumes universal-arrow-fo 𝔉 c r u
  shows ntcf-ua-fo α 𝔉 c r u :
    Hom_{O.Cα}𝔄(−,r) ↦_{CF.iso} Hom_{O.Cα}𝔅(−,c) ∘_{CF} op-cf 𝔉 :
    op-cat 𝔄 ↦↦_{Cα} cat-Set α
```

```
proof–
  from universal-arrow-foD[OF assms] have r: r ∈₀ 𝔄(|Obj|) and c: c ∈₀ 𝔅(|Obj|)
    by auto
  show ?thesis
    by
      (
        rule is-functor.cf-ntcf-ua-of-is-iso-ntcf
          [
            OF is-functor-op,
            unfolded cat-op-simps,
            OF assms,
            unfolded
              HomDom.cat-op-cat-cf-Hom-snd[OF r]
              HomCod.cat-op-cat-cf-Hom-snd[OF c]
              ntcf-ua-fo-def[symmetric]
          ]
      )
qed

lemmas [cat-cs-intros] = is-functor.cf-ntcf-ua-fo-is-iso-ntcf

lemma (in is-functor) cf-ua-of-if-ntcf-ua-of-is-iso-ntcf:
  assumes r ∈₀ 𝔄(|Obj|)
    and u : c ↦_𝔅 𝔉(|ObjMap|)(|r|)
    and ntcf-ua-of α 𝔉 c r u :
      Hom_{O.Cα}𝔄(r,−) ↦_{CF.iso} Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉 : 𝔄 ↦↦_{Cα} cat-Set α
  shows universal-arrow-of 𝔉 c r u
proof(rule universal-arrow-ofI)
  interpret ua-of-u: is-iso-ntcf
    α
    𝔄
    ‹cat-Set α›
    ‹Hom_{O.Cα}𝔄(r,−)›
    ‹Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉›
    ‹ntcf-ua-of α 𝔉 c r u›
    by (rule assms(3))
  fix r′ u′ assume prems: r′ ∈₀ 𝔄(|Obj|) u′ : c ↦_𝔅 𝔉(|ObjMap|)(|r′|)
  have ntcf-ua-of α 𝔉 c r u(|NTMap|)(|r′|) :
    Hom_{O.Cα}𝔄(r,−)(|ObjMap|)(|r′|) ↦_{iso cat-Set α}
    (Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉)(|ObjMap|)(|r′|)
    by (rule is-iso-ntcf.iso-ntcf-is-iso-arr[OF assms(3) prems(1)])
  from this is-functor-axioms assms(1–2) prems have uof-r′:
    umap-of 𝔉 c r u r′ : Hom 𝔄 r r′ ↦_{iso cat-Set α} Hom 𝔅 c (𝔉(|ObjMap|)(|r′|))
    by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros)
  note uof-r′ = cat-Set-is-iso-arrD[OF uof-r′]
  interpret uof-r′: v11 ‹umap-of 𝔉 c r u r′(|ArrVal|)› by (rule uof-r′(2))
  from
    uof-r′.v11-vrange-ex1-eq[
      THEN iffD1, unfolded uof-r′(3,4) in-Hom-iff, OF prems(2)
    ]
  show ∃!f′. f′ : r ↦_𝔄 r′ ∧ u′ = umap-of 𝔉 c r u r′(|ArrVal|)(|f′|)
    by metis
qed (intro assms)+

lemma (in is-functor) cf-ua-fo-if-ntcf-ua-fo-is-iso-ntcf:
  assumes r ∈₀ 𝔄(|Obj|)
    and u : 𝔉(|ObjMap|)(|r|) ↦_𝔅 c
    and ntcf-ua-fo α 𝔉 c r u :
```

```
    Hom_{O.Cα}𝔄(−,r) ↦_{CF.iso} Hom_{O.Cα}𝔅(−,c) ∘_{CF} op-cf 𝔉 :
    op-cat 𝔄 ↦↦_{Cα} cat-Set α
  shows universal-arrow-fo 𝔉 c r u
proof–
  from assms(2) have c: c ∈₀ 𝔅(|Obj|) by auto
  show ?thesis
    by
      (
        rule is-functor.cf-ua-of-if-ntcf-ua-of-is-iso-ntcf
          [
            OF is-functor-op,
            unfolded cat-op-simps,
            OF assms(1,2),
            unfolded
              HomDom.cat-op-cat-cf-Hom-snd[OF assms(1)]
              HomCod.cat-op-cat-cf-Hom-snd[OF c]
              ntcf-ua-fo-def[symmetric],
            OF assms(3)
          ]
      )
qed

lemma (in is-functor) cf-universal-arrow-of-if-is-iso-ntcf:
  assumes r ∈₀ 𝔄(|Obj|)
    and c ∈₀ 𝔅(|Obj|)
    and φ :
      Hom_{O.Cα}𝔄(r,−) ↦_{CF.iso} Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉 : 𝔄 ↦↦_{Cα} cat-Set α
  shows universal-arrow-of 𝔉 c r (φ(|NTMap|)(|r|)(|ArrVal|)(|𝔄(|CId|)(|r|)|))
    (is ‹universal-arrow-of 𝔉 c r ?u›)
proof–

  interpret φ: is-iso-ntcf
    α 𝔄 ‹cat-Set α› ‹Hom_{O.Cα}𝔄(r,−)› ‹Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉› φ
    by (rule assms(3))

  show ?thesis
  proof(intro universal-arrow-ofI assms)

    from assms(1,2) show u: ?u : c ↦_𝔅 𝔉(|ObjMap|)(|r|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
        )
    fix r′ u′ assume prems: r′ ∈₀ 𝔄(|Obj|) u′ : c ↦_𝔅 𝔉(|ObjMap|)(|r′|)
    have φr′-ArrVal-app[symmetric, cat-cs-simps]:
      φ(|NTMap|)(|r′|)(|ArrVal|)(|f′|) =
        𝔉(|ArrMap|)(|f′|) ∘_{A𝔅} φ(|NTMap|)(|r|)(|ArrVal|)(|𝔄(|CId|)(|r|)|)
      if f′ : r ↦_𝔄 r′ for f′
    proof–
      have φ(|NTMap|)(|r′|) ∘_{A cat-Set α} Hom_{O.Cα}𝔄(r,−)(|ArrMap|)(|f′|) =
        (Hom_{O.Cα}𝔅(c,−) ∘_{CF} 𝔉)(|ArrMap|)(|f′|) ∘_{A cat-Set α} φ(|NTMap|)(|r|)
        using that by (intro φ.ntcf-Comp-commute)
      then have
        φ(|NTMap|)(|r′|) ∘_{A cat-Set α} cf-hom 𝔄 [𝔄(|CId|)(|r|), f′]₀ =
          cf-hom 𝔅 [𝔅(|CId|)(|c|), 𝔉(|ArrMap|)(|f′|)]₀ ∘_{A cat-Set α} φ(|NTMap|)(|r|)
        using assms(1,2) that prems
        by
```

```
    (
      cs-prems cs-shallow
        cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
    )
  then have
    (φ⟦NTMap⟧⟦r'⟧ ∘_A cat-Set α
    cf-hom 𝔄 [𝔄⟦CId⟧⟦r⟧, f']∘)⟦ArrVal⟧⟦𝔄⟦CId⟧⟦r⟧⟧ =
      (cf-hom 𝔅 [𝔅⟦CId⟧⟦c⟧, 𝔉⟦ArrMap⟧⟦f'⟧]∘ ∘_A cat-Set α
      φ⟦NTMap⟧⟦r⟧)⟦ArrVal⟧⟦𝔄⟦CId⟧⟦r⟧⟧
     by simp
  from this assms(1,2) u that show ?thesis
    by
      (
      cs-prems cs-shallow
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed

show ∃!f'. f' : r ↦_𝔄 r' ∧ u' = umap-of 𝔉 c r ?u r'⟦ArrVal⟧⟦f'⟧
proof(intro ex1I conjI; (elim conjE)?)
  from assms prems show
    (φ⟦NTMap⟧⟦r'⟧)⁻¹_C cat-Set α⟦ArrVal⟧⟦u'⟧ : r ↦_𝔄 r'
    by
      (
      cs-concl
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro: cat-cs-intros cat-arrow-cs-intros
      )
  with assms(1,2) prems show u' =
    umap-of 𝔉 c r ?u r'⟦ArrVal⟧⟦(φ⟦NTMap⟧⟦r'⟧)⁻¹_C cat-Set α⟦ArrVal⟧⟦u'⟧⟧
    by
      (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro: cat-arrow-cs-intros cat-cs-intros cat-op-intros
      )
  fix f' assume prems':
    f' : r ↦_𝔄 r'
    u' = umap-of 𝔉 c r (φ⟦NTMap⟧⟦r⟧⟦ArrVal⟧⟦𝔄⟦CId⟧⟦r⟧⟧) r'⟦ArrVal⟧⟦f'⟧
  from prems'(2,1) assms(1,2) have u'-def:
    u' = 𝔉⟦ArrMap⟧⟦f'⟧ ∘_A𝔅 φ⟦NTMap⟧⟦r⟧⟦ArrVal⟧⟦𝔄⟦CId⟧⟦r⟧⟧
    by
      (
      cs-prems cs-shallow
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro: cat-cs-intros cat-op-intros
      )
  from prems' show f' = (φ⟦NTMap⟧⟦r'⟧)⁻¹_C cat-Set α⟦ArrVal⟧⟦u'⟧
    unfolding u'-def φr'-ArrVal-app[OF prems'(1)]
    by
      (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-arrow-cs-intros cat-cs-intros cat-op-intros
      )

qed
```

qed

qed

lemma (**in** *is-functor*) *cf-universal-arrow-fo-if-is-iso-ntcf*:
assumes $r \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $c \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
and φ :
$Hom_{O.C\alpha}\mathfrak{A}(-,r) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{B}(-,c) \circ_{CF} op\text{-}cf\ \mathfrak{F}$:
$op\text{-}cat\ \mathfrak{A} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
shows *universal-arrow-fo* $\mathfrak{F}\ c\ r\ (\varphi(\!|NTMap|\!)(\!|r|\!)(\!|ArrVal|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|r|\!)|\!))$
by
(
rule is-functor.cf-universal-arrow-of-if-is-iso-ntcf
[
OF is-functor-op,
unfolded cat-op-simps,
OF assms(*1*,*2*),
unfolded
HomDom.cat-op-cat-cf-Hom-snd[*OF assms*(*1*)]
HomCod.cat-op-cat-cf-Hom-snd[*OF assms*(*2*)]
ntcf-ua-fo-def[*symmetric*],
OF assms(*3*)
]
)

3 Limits and colimits

3.1 Background

named-theorems *cat-lim-cs-simps*
named-theorems *cat-lim-cs-intros*

3.2 Limit and colimit

3.2.1 Definition and elementary properties

The concept of a limit is introduced in Chapter III-4 in [9]; the concept of a colimit is introduced in Chapter III-3 in [9].

locale *is-cat-limit* = *is-cat-cone* α r $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u **for** α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u +
  assumes *cat-lim-ua-fo*: $\bigwedge u'\ r'.\ u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \Longrightarrow$
    $\exists! f'.\ f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f'$

syntax *-is-cat-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} <_{CF.lim} \text{-} :/ \text{-} \mapsto\mapsto_{C^1} \text{-})\rangle\ [51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-cat-limit* $\rightleftharpoons$ *is-cat-limit*
translations $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-cat-limit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

locale *is-cat-colimit* = *is-cat-cocone* α r $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u **for** α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u +
  assumes *cat-colim-ua-of*: $\bigwedge u'\ r'.\ u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \Longrightarrow$
    $\exists! f'.\ f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF} u$

syntax *-is-cat-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} >_{CF.colim} \text{-} :/ \text{-} \mapsto\mapsto_{C^1} \text{-})\rangle\ [51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-cat-colimit* $\rightleftharpoons$ *is-cat-colimit*
translations $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-cat-colimit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

Rules.

lemma (**in** *is-cat-limit*) *is-cat-limit-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
  shows $u : r' <_{CF.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-limit-axioms*)

mk-ide rf *is-cat-limit-def*[*unfolded is-cat-limit-axioms-def*]
  |*intro is-cat-limitI*|
  |*dest is-cat-limitD*[*dest*]|
  |*elim is-cat-limitE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-limitD*(*1*)

lemma (**in** *is-cat-colimit*) *is-cat-colimit-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
  shows $u : \mathfrak{F}' >_{CF.colim} r' : \mathfrak{J}' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-colimit-axioms*)

mk-ide rf *is-cat-colimit-def*[*unfolded is-cat-colimit-axioms-def*]
  |*intro is-cat-colimitI*|
  |*dest is-cat-colimitD*[*dest*]|
  |*elim is-cat-colimitE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-colimitD*(*1*)

Limits, colimits and universal arrows.

lemma (**in** *is-cat-limit*) *cat-lim-is-universal-arrow-fo*:
  universal-arrow-fo (Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u)
proof(*intro is-functor.universal-arrow-foI*)

  define β **where** $\beta = \alpha + \omega$
  have β: $\mathcal{Z}$ β **and** $\alpha\beta$: $\alpha \in_\circ \beta$
    by (*simp-all add*: β-*def* $\mathcal{Z}$-*Limit-*$\alpha\omega$ $\mathcal{Z}$-ω-$\alpha\omega$ $\mathcal{Z}$-*def* $\mathcal{Z}$-α-$\alpha\omega$)
  then interpret β: $\mathcal{Z}$ β **by** *simp*

  show Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}$: $\mathfrak{C}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}$
    by
      (
        intro
          β $\alpha\beta$
          cf-diagonal-is-functor
          NTDom.HomDom.category-axioms
          NTDom.HomCod.category-axioms
)

  show $r \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** (*intro cat-cone-obj*)
  then show *ntcf-arrow* u : Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|r|\!)$ $\mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* $\mathfrak{F}$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*
)

  fix r' u' **assume** *prems*:
    $r' \in_\circ \mathfrak{C}(\!|Obj|\!)$ u' : Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|r'|\!)$ $\mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* $\mathfrak{F}$
  from *prems*(*1*) **have** [*cat-cs-simps*]:
    cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* $\mathfrak{F}$) = $\mathfrak{F}$
    cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r')) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r'
    by (*cs-concl* **cs-simp**: *cat-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros*)+
  from *prems*(*2*,*1*) **have**
    u' : *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r') $\mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* $\mathfrak{F}$
    by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps*)
  note u'[*unfolded cat-cs-simps*] = *cat-FUNCT-is-arrD*[*OF this*]

  from *cat-lim-ua-fo*[*OF is-cat-coneI*[*OF* u'(*1*) *prems*(*1*)]] **obtain** f
    where f: f : r' $\mapsto_{\mathfrak{C}}$ r
      and [*symmetric*, *cat-cs-simps*]:
        ntcf-of-ntcf-arrow $\mathfrak{J}$ $\mathfrak{C}$ u' = u $\bullet_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f
      and *f-unique*:
        ⟦
          f' : r' $\mapsto_{\mathfrak{C}}$ r;
          ntcf-of-ntcf-arrow $\mathfrak{J}$ $\mathfrak{C}$ u' = u $\bullet_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f'
        ⟧ ⟹ f' = f
    for f'
    by *metis*

  show $\exists!f'$.
    f' : r' $\mapsto_{\mathfrak{C}}$ r ∧
    u' = *umap-fo* (Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u) $r'(\!|ArrVal|\!)(\!|f'|\!)$
  proof(*intro ex1I conjI*; (*elim conjE*)*?*)
    show f : r' $\mapsto_{\mathfrak{C}}$ r **by** (*rule f*)
    with $\alpha\beta$ *cat-cone-obj* **show** u'-*def*:
      u' = *umap-fo* (Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u) $r'(\!|ArrVal|\!)(\!|f|\!)$

```
    by
      (
        cs-concl
          cs-simp: u'(2)[symmetric] cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
    fix f' assume prems':
      f' : r' ↦_ℭ r
      u' = umap-fo (Δ_CF α ℑ ℭ) (cf-map ℱ) r (ntcf-arrow u) r'(|ArrVal|)(|f'|)
    from prems'(2) αβ f prems' cat-cone-obj have u'-def':
      u' = ntcf-arrow (u ∘_NTCF ntcf-const ℑ ℭ f')
      by
        (
        cs-prems
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
    from prems'(1) have ntcf-of-ntcf-arrow ℑ ℭ u' = u ∘_NTCF ntcf-const ℑ ℭ f'
      by
        (
        cs-concl
          cs-simp: cat-FUNCT-cs-simps u'-def' cs-intro: cat-cs-intros
        )
    from f-unique[OF prems'(1) this] show f' = f .

  qed

qed

lemma (in is-cat-cone) cat-cone-is-cat-limit:
  assumes universal-arrow-fo (Δ_CF α ℑ ℭ) (cf-map ℱ) c (ntcf-arrow 𝔑)
  shows 𝔑 : c <_CF.lim ℱ : ℑ ↦↦_Cα ℭ
proof–

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝒵-Limit-αω 𝒵-ω-αω 𝒵-def 𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(intro is-cat-limitI is-cat-cone-axioms)
    fix u' c' assume prems: u' : c' <_CF.cone ℱ : ℑ ↦↦_Cα ℭ

    interpret u': is-cat-cone α c' ℑ ℭ ℱ u' by (rule prems)

    from u'.cat-cone-obj have u'-is-arr:
      ntcf-arrow u' : Δ_CF α ℑ ℭ(|ObjMap|)(|c'|) ↦_cat-FUNCT α ℑ ℭ cf-map ℱ
      by
        (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )

    from is-functor.universal-arrow-foD(3)
      [
        OF
          cf-diagonal-is-functor[
            OF β αβ NTDom.HomDom.category-axioms NTDom.HomCod.category-axioms
```

```
        ]
      assms
      u'.cat-cone-obj
      u'-is-arr
    ]
  obtain f where f: f : c' ↦_ℭ c
    and u'-def': ntcf-arrow u' =
      umap-fo (Δ_CF α ℑ ℭ) (cf-map ℱ) c (ntcf-arrow ℜ) c'(|ArrVal|)(|f|)
    and f'-unique:
      [[
        f' : c' ↦_ℭ c;
        ntcf-arrow u' =
          umap-fo (Δ_CF α ℑ ℭ) (cf-map ℱ) c (ntcf-arrow ℜ) c'(|ArrVal|)(|f'|)
      ]] ⟹ f' = f
    for f'
    by metis

  from u'-def' αβ f cat-cone-obj have u'-def:
    u' = ℜ ∘_NTCF ntcf-const ℑ ℭ f
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )

  show ∃!f'. f' : c' ↦_ℭ c ∧ u' = ℜ ∘_NTCF ntcf-const ℑ ℭ f'
  proof(intro ex1I conjI; (elim conjE)?, (rule f)?, (rule u'-def)?)
    fix f'' assume prems':
      f'' : c' ↦_ℭ c u' = ℜ ∘_NTCF ntcf-const ℑ ℭ f''
    from αβ prems' have
      ntcf-arrow u' =
        umap-fo (Δ_CF α ℑ ℭ) (cf-map ℱ) c (ntcf-arrow ℜ) c'(|ArrVal|)(|f''|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
    from f'-unique[OF prems'(1) this] show f'' = f.
  qed

qed

lemma (in is-cat-colimit) cat-colim-is-universal-arrow-of:
  universal-arrow-of (Δ_CF α ℑ ℭ) (cf-map ℱ) r (ntcf-arrow u)
proof(intro is-functor.universal-arrow-ofI)

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝒵-Limit-αω 𝒵-ω-αω 𝒵-def 𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show Δ_CF α ℑ ℭ : ℭ ↦↦_Cβ cat-FUNCT α ℑ ℭ
    by
      (
```

```
      intro
        β αβ
        cf-diagonal-is-functor
        NTDom.HomDom.category-axioms
        NTDom.HomCod.category-axioms
    )

show r ∈∘ 𝔈(|Obj|) by (intro cat-cocone-obj)

then show ntcf-arrow u : cf-map 𝔉 ↦_cat-FUNCT α 𝔍 𝔈 Δ_CF α 𝔍 𝔈(|ObjMap|)(|r|)
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
    )

fix r′ u′ assume prems:
  r′ ∈∘ 𝔈(|Obj|) u′ : cf-map 𝔉 ↦_cat-FUNCT α 𝔍 𝔈 Δ_CF α 𝔍 𝔈(|ObjMap|)(|r′|)
from prems(1) have [cat-cs-simps]:
  cf-of-cf-map 𝔍 𝔈 (cf-map 𝔉) = 𝔉
  cf-of-cf-map 𝔍 𝔈 (cf-map (cf-const 𝔍 𝔈 r′)) = cf-const 𝔍 𝔈 r′
  by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)+
from prems(2,1) have
  u′ : cf-map 𝔉 ↦_cat-FUNCT α 𝔍 𝔈 cf-map (cf-const 𝔍 𝔈 r′)
  by (cs-prems cs-shallow cs-simp: cat-cs-simps)
note u′[unfolded cat-cs-simps] = cat-FUNCT-is-arrD[OF this]

from cat-colim-ua-of[OF is-cat-coconeI[OF u′(1) prems(1)]] obtain f
  where f: f : r ↦_𝔈 r′
    and [symmetric, cat-cs-simps]:
      ntcf-of-ntcf-arrow 𝔍 𝔈 u′ = ntcf-const 𝔍 𝔈 f ∘_NTCF u
    and f-unique:
      ⟦
        f′ : r ↦_𝔈 r′;
        ntcf-of-ntcf-arrow 𝔍 𝔈 u′ = ntcf-const 𝔍 𝔈 f′ ∘_NTCF u
      ⟧ ⟹ f′ = f
  for f′
  by metis

show ∃!f′.
  f′ : r ↦_𝔈 r′ ∧
  u′ = umap-of (Δ_CF α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u) r′(|ArrVal|)(|f′|)
proof(intro ex1I conjI; (elim conjE)?)

  show f : r ↦_𝔈 r′ by (rule f)
  with αβ cat-cocone-obj show u′-def:
    u′ = umap-of (Δ_CF α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u) r′(|ArrVal|)(|f|)
    by
      (
        cs-concl
          cs-simp: u′(2)[symmetric] cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )

  fix f′ assume prems′:
    f′ : r ↦_𝔈 r′
    u′ = umap-of (Δ_CF α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u) r′(|ArrVal|)(|f′|)
  from prems′(2) αβ f prems′ cat-cocone-obj have u′-def′:
```

```
    u′ = ntcf-arrow (ntcf-const 𝔍 ℭ f′ ∘_NTCF u)
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  from prems′(1) have ntcf-of-ntcf-arrow 𝔍 ℭ u′ = ntcf-const 𝔍 ℭ f′ ∘_NTCF u
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-cs-simps u′-def′ cs-intro: cat-cs-intros
      )
  from f-unique[OF prems′(1) this] show f′ = f .

qed

qed

lemma (in is-cat-cocone) cat-cocone-is-cat-colimit:
  assumes universal-arrow-of (Δ_CF α 𝔍 ℭ) (cf-map 𝔉) c (ntcf-arrow 𝔑)
  shows 𝔑 : 𝔉 >_CF.colim c : 𝔍 ↦↦_Cα ℭ
proof–

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝒵-Limit-αω 𝒵-ω-αω 𝒵-def 𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(intro is-cat-colimitI is-cat-cocone-axioms)

    fix u′ c′ assume prems: u′ : 𝔉 >_CF.cocone c′ : 𝔍 ↦↦_Cα ℭ

    interpret u′: is-cat-cocone α c′ 𝔍 ℭ 𝔉 u′ by (rule prems)

    from u′.cat-cocone-obj have u′-is-arr:
      ntcf-arrow u′ : cf-map 𝔉 ↦_cat-FUNCT α 𝔍 ℭ Δ_CF α 𝔍 ℭ⦇ObjMap⦈⦇c′⦈
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )

    from is-functor.universal-arrow-ofD(3)
      [
        OF
          cf-diagonal-is-functor[
            OF β αβ NTDom.HomDom.category-axioms NTDom.HomCod.category-axioms
            ]
          assms
          u′.cat-cocone-obj
          u′-is-arr
      ]
    obtain f where f: f : c ↦_ℭ c′
      and u′-def′: ntcf-arrow u′ =
        umap-of (Δ_CF α 𝔍 ℭ) (cf-map 𝔉) c (ntcf-arrow 𝔑) c′⦇ArrVal⦈⦇f⦈
      and f′-unique:
```

```
      ⟦
        f' : c ↦_𝔈 c';
        ntcf-arrow u' =
          umap-of (Δ_CF α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(|ArrVal|)(|f'|)
      ⟧ ⟹ f' = f
    for f'
    by metis

  from u'-def' αβ f cat-cocone-obj have u'-def:
    u' = ntcf-const 𝔍 𝔈 f ∘_NTCF 𝔑
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )

  show ∃!f'. f' : c ↦_𝔈 c' ∧ u' = ntcf-const 𝔍 𝔈 f' ∘_NTCF 𝔑
  proof(intro ex1I conjI; (elim conjE)?, (rule f)?, (rule u'-def)?)
    fix f'' assume prems':
      f'' : c ↦_𝔈 c' u' = ntcf-const 𝔍 𝔈 f'' ∘_NTCF 𝔑
    from αβ prems' have
      ntcf-arrow u' =
        umap-of (Δ_CF α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(|ArrVal|)(|f''|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
    from f'-unique[OF prems'(1) this] show f'' = f.
  qed

 qed

qed
```

Duality.

```
lemma (in is-cat-limit) is-cat-colimit-op:
  op-ntcf u : op-cf 𝔉 >_CF.colim r : op-cat 𝔍 ↦↦_Cα op-cat 𝔈
proof(intro is-cat-colimitI)
  show op-ntcf u : op-cf 𝔉 >_CF.cocone r : op-cat 𝔍 ↦↦_Cα op-cat 𝔈
    by (cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros)
  fix u' r' assume prems:
    u' : op-cf 𝔉 >_CF.cocone r' : op-cat 𝔍 ↦↦_Cα op-cat 𝔈
  interpret u': is-cat-cocone α r' ‹op-cat 𝔍› ‹op-cat 𝔈› ‹op-cf 𝔉› u'
    by (rule prems)
  from cat-lim-ua-fo[OF u'.is-cat-cone-op[unfolded cat-op-simps]] obtain f
    where f: f : r' ↦_𝔈 r
      and op-u'-def: op-ntcf u' = u ∘_NTCF ntcf-const 𝔍 𝔈 f
      and f-unique:
        ⟦ f' : r' ↦_𝔈 r; op-ntcf u' = u ∘_NTCF ntcf-const 𝔍 𝔈 f' ⟧ ⟹
          f' = f
    for f'
    by metis
  from op-u'-def have op-ntcf (op-ntcf u') = op-ntcf (u ∘_NTCF ntcf-const 𝔍 𝔈 f)
    by simp
  from this f have u'-def:
```

```
    u' = ntcf-const (op-cat ℑ) (op-cat 𝔈) f ∘_NTCF op-ntcf u
    by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros)
  show ∃!f'.
    f' : r ↦_op-cat 𝔈 r' ∧
    u' = ntcf-const (op-cat ℑ) (op-cat 𝔈) f' ∘_NTCF op-ntcf u
  proof(intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?)
    fix f' assume prems':
      f' : r' ↦_𝔈 r
      u' = ntcf-const (op-cat ℑ) (op-cat 𝔈) f' ∘_NTCF op-ntcf u
    from prems'(2) have
      op-ntcf u' = op-ntcf (ntcf-const (op-cat ℑ) (op-cat 𝔈) f' ∘_NTCF op-ntcf u)
      by simp
    from this prems'(1) have op-ntcf u' = u ∘_NTCF ntcf-const ℑ 𝔈 f'
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
        )
    from f-unique[OF prems'(1) this] show f' = f.
  qed (intro u'-def f)+
qed

lemma (in is-cat-limit) is-cat-colimit-op'[cat-op-intros]:
  assumes 𝔉' = op-cf 𝔉 and ℑ' = op-cat ℑ and 𝔈' = op-cat 𝔈
  shows op-ntcf u : 𝔉' >_CF.colim r : ℑ' ↦↦_Cα 𝔈'
  unfolding assms by (rule is-cat-colimit-op)

lemmas [cat-op-intros] = is-cat-limit.is-cat-colimit-op'

lemma (in is-cat-colimit) is-cat-limit-op:
  op-ntcf u : r <_CF.lim op-cf 𝔉 : op-cat ℑ ↦↦_Cα op-cat 𝔈
proof(intro is-cat-limitI)
  show op-ntcf u : r <_CF.cone op-cf 𝔉 : op-cat ℑ ↦↦_Cα op-cat 𝔈
    by (cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros)
  fix u' r' assume prems:
    u' : r' <_CF.cone op-cf 𝔉 : op-cat ℑ ↦↦_Cα op-cat 𝔈
  interpret u': is-cat-cone α r' ‹op-cat ℑ› ‹op-cat 𝔈› ‹op-cf 𝔉› u'
    by (rule prems)
  from cat-colim-ua-of[OF u'.is-cat-cocone-op[unfolded cat-op-simps]] obtain f
    where f: f : r ↦_𝔈 r'
      and op-u'-def: op-ntcf u' = ntcf-const ℑ 𝔈 f ∘_NTCF u
      and f-unique:
        ⟦ f' : r ↦_𝔈 r'; op-ntcf u' = ntcf-const ℑ 𝔈 f' ∘_NTCF u ⟧ ⟹
          f' = f
    for f'
    by metis
  from op-u'-def have op-ntcf (op-ntcf u') = op-ntcf (ntcf-const ℑ 𝔈 f ∘_NTCF u)
    by simp
  from this f have u'-def:
    u' = op-ntcf u ∘_NTCF ntcf-const (op-cat ℑ) (op-cat 𝔈) f
    by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros)
  show ∃!f'.
    f' : r' ↦_op-cat 𝔈 r ∧
    u' = op-ntcf u ∘_NTCF ntcf-const (op-cat ℑ) (op-cat 𝔈) f'
  proof(intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?)
    fix f' assume prems':
      f' : r ↦_𝔈 r'
```

```
    u′ = op-ntcf u ∙_NTCF ntcf-const (op-cat 𝔍) (op-cat ℭ) f′
  from prems′(2) have
    op-ntcf u′ = op-ntcf (op-ntcf u ∙_NTCF ntcf-const (op-cat 𝔍) (op-cat ℭ) f′)
    by simp
  from this prems′(1) have op-ntcf u′ = ntcf-const 𝔍 ℭ f′ ∙_NTCF u
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
  from f-unique[OF prems′(1) this] show f′ = f.
 qed (intro u′-def f)+
qed

lemma (in is-cat-colimit) is-cat-colimit-op′[cat-op-intros]:
 assumes 𝔉′ = op-cf 𝔉 and 𝔍′ = op-cat 𝔍 and ℭ′ = op-cat ℭ
 shows op-ntcf u : r <_CF.lim 𝔉′ : 𝔍′ ↦↦_Cα ℭ′
 unfolding assms by (rule is-cat-limit-op)

lemmas [cat-op-intros] = is-cat-colimit.is-cat-colimit-op′
```

3.2.2 Universal property

```
lemma (in is-cat-limit) cat-lim-unique-cone′:
 assumes u′ : r′ <_CF.cone 𝔉 : 𝔍 ↦↦_Cα ℭ
 shows
   ∃!f′. f′ : r′ ↦_ℭ r ∧ (∀ j∈₀𝔍(|Obj|). u′(|NTMap|)(|j|) = u(|NTMap|)(|j|) ∘_Aℭ f′)
 by (fold helper-cat-cone-Comp-ntcf-vcomp-iff[OF assms(1)])
   (intro cat-lim-ua-fo assms)

lemma (in is-cat-limit) cat-lim-unique:
 assumes u′ : r′ <_CF.lim 𝔉 : 𝔍 ↦↦_Cα ℭ
 shows ∃!f′. f′ : r′ ↦_ℭ r ∧ u′ = u ∙_NTCF ntcf-const 𝔍 ℭ f′
 by (intro cat-lim-ua-fo[OF is-cat-limitD(1)[OF assms]])

lemma (in is-cat-limit) cat-lim-unique′:
 assumes u′ : r′ <_CF.lim 𝔉 : 𝔍 ↦↦_Cα ℭ
 shows
   ∃!f′. f′ : r′ ↦_ℭ r ∧ (∀ j∈₀𝔍(|Obj|). u′(|NTMap|)(|j|) = u(|NTMap|)(|j|) ∘_Aℭ f′)
 by (intro cat-lim-unique-cone′[OF is-cat-limitD(1)[OF assms]])

lemma (in is-cat-colimit) cat-colim-unique-cocone:
 assumes u′ : 𝔉 >_CF.cocone r′ : 𝔍 ↦↦_Cα ℭ
 shows ∃!f′. f′ : r ↦_ℭ r′ ∧ u′ = ntcf-const 𝔍 ℭ f′ ∙_NTCF u
proof–
 interpret u′: is-cat-cocone α r′ 𝔍 ℭ 𝔉 u′ by (rule assms(1))
 from u′.cat-cocone-obj have op-r′: r′ ∈₀ op-cat ℭ(|Obj|)
   unfolding cat-op-simps by simp
 from
   is-cat-limit.cat-lim-ua-fo[
     OF is-cat-limit-op u′.is-cat-cone-op, folded op-ntcf-ntcf-const
     ]
 obtain f′ where f′: f′ : r′ ↦_op-cat ℭ r
   and [cat-cs-simps]:
     op-ntcf u′ = op-ntcf u ∙_NTCF op-ntcf (ntcf-const 𝔍 ℭ f′)
   and unique:
     [[
```

```
      f'' : r' ↦_{op-cat 𝔈} r;
      op-ntcf u' = op-ntcf u ∙_{NTCF} op-ntcf (ntcf-const 𝔍 𝔈 f'')
    ⟧ ⟹ f'' = f'
  for f''
  by metis
 show ?thesis
 proof(intro ex1I conjI; (elim conjE)?)
  from f' show f': f' : r ↦_𝔈 r' unfolding cat-op-simps by simp
  show u' = ntcf-const 𝔍 𝔈 f' ∙_{NTCF} u
   by (rule eq-op-ntcf-iff[THEN iffD1], insert f')
     (cs-concl cs-intro: cat-cs-intros cs-simp: cat-cs-simps cat-op-simps)+
  fix f'' assume prems: f'' : r ↦_𝔈 r' u' = ntcf-const 𝔍 𝔈 f'' ∙_{NTCF} u
  from prems(1) have f'' : r' ↦_{op-cat 𝔈} r unfolding cat-op-simps by simp
  moreover from prems(1) have
   op-ntcf u' = op-ntcf u ∙_{NTCF} op-ntcf (ntcf-const 𝔍 𝔈 f'')
   unfolding prems(2)
   by (cs-concl cs-intro: cat-cs-intros cs-simp: cat-cs-simps cat-op-simps)
  ultimately show f'' = f' by (rule unique)
 qed
qed

lemma (in is-cat-colimit) cat-colim-unique-cocone':
 assumes u' : 𝔉 >_{CF.cocone} r' : 𝔍 ↦↦_{Cα} 𝔈
 shows
  ∃!f'. f' : r ↦_𝔈 r' ∧ (∀ j∈_∘ 𝔍⦇Obj⦈. u'⦇NTMap⦈⦇j⦈ = f' ∘_{A𝔈} u⦇NTMap⦈⦇j⦈)
 by (fold helper-cat-cocone-Comp-ntcf-vcomp-iff[OF assms(1)])
  (intro cat-colim-unique-cocone assms)

lemma (in is-cat-colimit) cat-colim-unique:
 assumes u' : 𝔉 >_{CF.colim} r' : 𝔍 ↦↦_{Cα} 𝔈
 shows ∃!f'. f' : r ↦_𝔈 r' ∧ u' = ntcf-const 𝔍 𝔈 f' ∙_{NTCF} u
 by (intro cat-colim-unique-cocone[OF is-cat-colimitD(1)[OF assms]])

lemma (in is-cat-colimit) cat-colim-unique':
 assumes u' : 𝔉 >_{CF.colim} r' : 𝔍 ↦↦_{Cα} 𝔈
 shows
  ∃!f'. f' : r ↦_𝔈 r' ∧ (∀ j∈_∘ 𝔍⦇Obj⦈. u'⦇NTMap⦈⦇j⦈ = f' ∘_{A𝔈} u⦇NTMap⦈⦇j⦈)
proof–
 interpret u': is-cat-colimit α 𝔍 𝔈 𝔉 r' u' by (rule assms(1))
 show ?thesis
  by (fold helper-cat-cocone-Comp-ntcf-vcomp-iff[OF u'.is-cat-cocone-axioms])
   (intro cat-colim-unique assms)
qed

lemma cat-lim-ex-is-iso-arr:
 assumes u : r <_{CF.lim} 𝔉 : 𝔍 ↦↦_{Cα} 𝔈 and u' : r' <_{CF.lim} 𝔉 : 𝔍 ↦↦_{Cα} 𝔈
 obtains f where f : r' ↦_{iso𝔈} r and u' = u ∙_{NTCF} ntcf-const 𝔍 𝔈 f
proof–
 interpret u: is-cat-limit α 𝔍 𝔈 𝔉 r u by (rule assms(1))
 interpret u': is-cat-limit α 𝔍 𝔈 𝔉 r' u' by (rule assms(2))
 define β where β = α + ω
 have β: 𝒵 β and αβ: α ∈_∘ β
  by (simp-all add: β-def u.𝒵-Limit-αω u.𝒵-ω-αω 𝒵-def u.𝒵-α-αω)
 then interpret β: 𝒵 β by simp
 have Δ: Δ_{CF} α 𝔍 𝔈 : 𝔈 ↦↦_{Cβ} cat-FUNCT α 𝔍 𝔈
  by
   (
    intro
```

```
      β αβ
      cf-diagonal-is-functor
      u.NTDom.HomDom.category-axioms
      u.NTDom.HomCod.category-axioms
    )
  then interpret Δ: is-functor β ℭ ‹cat-FUNCT α 𝔍 ℭ› ‹Δ_CF α 𝔍 ℭ› by simp
  from is-functor.cf-universal-arrow-fo-ex-is-iso-arr[
    OF Δ u.cat-lim-is-universal-arrow-fo u'.cat-lim-is-universal-arrow-fo
    ]
  obtain f where f: f : r' ↦iso_ℭ r
    and u': ntcf-arrow u' =
    umap-fo (Δ_CF α 𝔍 ℭ) (cf-map 𝔉) r (ntcf-arrow u) r'⦇ArrVal⦈⦇f⦈
    by auto
  from f have f : r' ↦_ℭ r by auto
  from u' this have u' = u ∙_NTCF ntcf-const 𝔍 ℭ f
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  with f that show ?thesis by simp
qed

lemma cat-lim-ex-is-iso-arr':
  assumes u : r <_CF.lim 𝔉 : 𝔍 ↦↦_Cα ℭ and u' : r' <_CF.lim 𝔉 : 𝔍 ↦↦_Cα ℭ
  obtains f where f : r' ↦iso_ℭ r
    and ⋀j. j ∈∘ 𝔍⦇Obj⦈ ⟹ u'⦇NTMap⦈⦇j⦈ = u⦇NTMap⦈⦇j⦈ ∘_Aℭ f
proof–
  interpret u: is-cat-limit α 𝔍 ℭ 𝔉 r u by (rule assms(1))
  interpret u': is-cat-limit α 𝔍 ℭ 𝔉 r' u' by (rule assms(2))
  from assms obtain f
    where iso-f: f : r' ↦iso_ℭ r and u'-def: u' = u ∙_NTCF ntcf-const 𝔍 ℭ f
    by (rule cat-lim-ex-is-iso-arr)
  then have f: f : r' ↦_ℭ r by auto
  then have u'⦇NTMap⦈⦇j⦈ = u⦇NTMap⦈⦇j⦈ ∘_Aℭ f if j ∈∘ 𝔍⦇Obj⦈ for j
    by
      (
        intro u.helper-cat-cone-ntcf-vcomp-Comp[
          OF u'.is-cat-cone-axioms f u'-def that
          ]
      )
  with iso-f that show ?thesis by simp
qed

lemma cat-colim-ex-is-iso-arr:
  assumes u : 𝔉 >_CF.colim r : 𝔍 ↦↦_Cα ℭ
    and u' : 𝔉 >_CF.colim r' : 𝔍 ↦↦_Cα ℭ
  obtains f where f : r ↦iso_ℭ r' and u' = ntcf-const 𝔍 ℭ f ∙_NTCF u
proof–
  interpret u: is-cat-colimit α 𝔍 ℭ 𝔉 r u by (rule assms(1))
  interpret u': is-cat-colimit α 𝔍 ℭ 𝔉 r' u' by (rule assms(2))
  obtain f where f: f : r' ↦iso_op-cat ℭ r
    and [cat-cs-simps]:
      op-ntcf u' = op-ntcf u ∙_NTCF ntcf-const (op-cat 𝔍) (op-cat ℭ) f
    by
      (
        elim cat-lim-ex-is-iso-arr[
```

```
      OF u.is-cat-limit-op u'.is-cat-limit-op
      ]
  )
 from f have iso-f: f : r ↦_iso_𝔈 r' unfolding cat-op-simps by simp
 then have f: f : r ↦_𝔈 r' by auto
 have u' = ntcf-const 𝔍 𝔈 f ∙_NTCF u
  by (rule eq-op-ntcf-iff[THEN iffD1], insert f)
   (cs-concl cs-intro: cat-cs-intros cs-simp: cat-cs-simps cat-op-simps)+
 from iso-f this that show ?thesis by simp
qed

lemma cat-colim-ex-is-iso-arr':
 assumes u : 𝔉 >_CF.colim r : 𝔍 ↦↦_Cα 𝔈
  and u' : 𝔉 >_CF.colim r' : 𝔍 ↦↦_Cα 𝔈
 obtains f where f : r ↦_iso_𝔈 r'
  and ⋀j. j ∈₀ 𝔍(|Obj|) ⟹ u'(|NTMap|)(|j|) = f ∘_A𝔈 u(|NTMap|)(|j|)
proof–
 interpret u: is-cat-colimit α 𝔍 𝔈 𝔉 r u by (rule assms(1))
 interpret u': is-cat-colimit α 𝔍 𝔈 𝔉 r' u' by (rule assms(2))
 from assms obtain f
  where iso-f: f : r ↦_iso_𝔈 r' and u'-def: u' = ntcf-const 𝔍 𝔈 f ∙_NTCF u
  by (rule cat-colim-ex-is-iso-arr)
 then have f: f : r ↦_𝔈 r' by auto
 then have u'(|NTMap|)(|j|) = f ∘_A𝔈 u(|NTMap|)(|j|) if j ∈₀ 𝔍(|Obj|) for j
  by
   (
    intro u.helper-cat-cocone-ntcf-vcomp-Comp[
     OF u'.is-cat-cocone-axioms f u'-def that
     ]
   )
 with iso-f that show ?thesis by simp
qed
```

3.2.3 Further properties

```
lemma (in is-cat-limit) cat-lim-is-cat-limit-if-is-iso-arr:
 assumes f : r' ↦_iso_𝔈 r
 shows u ∙_NTCF ntcf-const 𝔍 𝔈 f : r' <_CF.lim 𝔉 : 𝔍 ↦↦_Cα 𝔈
proof–
 note f = is-iso-arrD(1)[OF assms(1)]
 from f(1) interpret u': is-cat-cone α r' 𝔍 𝔈 𝔉 ‹u ∙_NTCF ntcf-const 𝔍 𝔈 f›
  by (cs-concl cs-intro: cat-lim-cs-intros cat-cs-intros)
 define β where β = α + ω
 have β: 𝒵 β and αβ: α ∈₀ β
  by (simp-all add: β-def 𝒵-Limit-αω 𝒵-ω-αω 𝒵-def 𝒵-α-αω)
 then interpret β: 𝒵 β by simp
 show ?thesis
 proof
  (
   intro u'.cat-cone-is-cat-limit,
   rule is-functor.universal-arrow-fo-if-universal-arrow-fo,
   rule cf-diagonal-is-functor[OF β αβ],
   rule NTDom.HomDom.category-axioms,
   rule NTDom.HomCod.category-axioms,
   rule cat-lim-is-universal-arrow-fo
  )
  show f : r' ↦_iso_𝔈 r by (rule assms(1))
  from αβ f show
```

```
    ntcf-arrow (u ∘NTCF ntcf-const 𝔍 ℭ f) =
      umap-fo (ΔCF α 𝔍 ℭ) (cf-map 𝔉) r (ntcf-arrow u) r′(|ArrVal|)(|f|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  qed
qed

lemma (in is-cat-colimit) cat-colim-is-cat-colimit-if-is-iso-arr:
  assumes f : r ↦iso ℭ r′
  shows ntcf-const 𝔍 ℭ f ∘NTCF u : 𝔉 >CF.colim r′ : 𝔍 ↦↦Cα ℭ
proof–
  note f = is-iso-arrD[OF assms(1)]
  from f(1) interpret u′: is-cat-cocone α r′ 𝔍 ℭ 𝔉 ‹ntcf-const 𝔍 ℭ f ∘NTCF u›
    by (cs-concl cs-intro: cat-lim-cs-intros cat-cs-intros)
  from f have [symmetric, cat-op-simps]:
    op-ntcf (ntcf-const 𝔍 ℭ f ∘NTCF u) =
      op-ntcf u ∘NTCF ntcf-const (op-cat 𝔍) (op-cat ℭ) f
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-op-intros
      )
  show ?thesis
    by
      (
        rule is-cat-limit.is-cat-colimit-op
          [
            OF is-cat-limit.cat-lim-is-cat-limit-if-is-iso-arr[
              OF is-cat-limit-op, unfolded cat-op-simps, OF assms(1)
              ],
            unfolded cat-op-simps
          ]
      )
qed

lemma ntcf-cf-comp-is-cat-limit-if-is-iso-functor:
  assumes u : r <CF.lim 𝔉 : 𝔅 ↦↦Cα ℭ and 𝔊 : 𝔄 ↦↦C.isoα 𝔅
  shows u ∘NTCF−CF 𝔊 : r <CF.lim 𝔉 ∘CF 𝔊 : 𝔄 ↦↦Cα ℭ
proof(intro is-cat-limitI)
  interpret u: is-cat-limit α 𝔅 ℭ 𝔉 r u by (rule assms(1))
  interpret 𝔊: is-iso-functor α 𝔄 𝔅 𝔊 by (rule assms(2))
  note [cf-cs-simps] = is-iso-functor-is-iso-arr(2,3)
  show u𝔊: u ∘NTCF−CF 𝔊 : r <CF.cone 𝔉 ∘CF 𝔊 : 𝔄 ↦↦Cα ℭ
    by (intro is-cat-coneI)
      (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  fix u′ r′ assume prems: u′ : r′ <CF.cone 𝔉 ∘CF 𝔊 : 𝔄 ↦↦Cα ℭ
  then interpret u′: is-cat-cone α r′ 𝔄 ℭ ‹𝔉 ∘CF 𝔊› u′ by simp
  have u′ ∘NTCF−CF inv-cf 𝔊 : r′ <CF.cone 𝔉 : 𝔅 ↦↦Cα ℭ
    by (intro is-cat-coneI)
      (
        cs-concl
          cs-simp: cat-cs-simps cf-cs-simps
          cs-intro: cat-cs-intros cf-cs-intros
      )
```

```
  from is-cat-limit.cat-lim-ua-fo[OF assms(1) this] obtain f
    where f: f : r′ ↦_ℭ r
      and u′-𝔊: u′ ∘_NTCF-CF inv-cf 𝔊 = u ⋆_NTCF ntcf-const 𝔅 ℭ f
      and f′f:
        ⟦
          f′ : r′ ↦_ℭ r;
          u′ ∘_NTCF-CF inv-cf 𝔊 = u ⋆_NTCF ntcf-const 𝔅 ℭ f′
        ⟧ ⟹ f′ = f
    for f′
    by metis
  from u′-𝔊 have u′-inv𝔊-𝔊:
    (u′ ∘_NTCF-CF inv-cf 𝔊) ∘_NTCF-CF 𝔊 = (u ⋆_NTCF ntcf-const 𝔅 ℭ f) ∘_NTCF-CF 𝔊
    by simp
  show ∃!f′. f′ : r′ ↦_ℭ r ∧ u′ = u ∘_NTCF-CF 𝔊 ⋆_NTCF ntcf-const 𝔄 ℭ f′
  proof(intro ex1I conjI; (elim conjE)?)
    show f : r′ ↦_ℭ r by (rule f)
    from u′-inv𝔊-𝔊 f show u′ = u ∘_NTCF-CF 𝔊 ⋆_NTCF ntcf-const 𝔄 ℭ f
      by
        (
          cs-prems
            cs-simp:
              cf-cs-simps cat-cs-simps
              ntcf-cf-comp-ntcf-cf-comp-assoc
              ntcf-vcomp-ntcf-cf-comp[symmetric]
            cs-intro: cat-cs-intros cf-cs-intros
        )
    fix f′ assume prems:
      f′ : r′ ↦_ℭ r u′ = u ∘_NTCF-CF 𝔊 ⋆_NTCF ntcf-const 𝔄 ℭ f′
    from prems(2) have
      u′ ∘_NTCF-CF inv-cf 𝔊 =
        (u ∘_NTCF-CF 𝔊 ⋆_NTCF ntcf-const 𝔄 ℭ f′) ∘_NTCF-CF inv-cf 𝔊
      by simp
    from this f prems(1) have u′ ∘_NTCF-CF inv-cf 𝔊 = u ⋆_NTCF ntcf-const 𝔅 ℭ f′
      by
        (
          cs-prems
            cs-simp:
              cat-cs-simps cf-cs-simps
              ntcf-vcomp-ntcf-cf-comp[symmetric]
              ntcf-cf-comp-ntcf-cf-comp-assoc
            cs-intro: cf-cs-intros cat-cs-intros
        )
    then show f′ = f by (intro f′f prems(1))
  qed
qed

lemma ntcf-cf-comp-is-cat-limit-if-is-iso-functor′[cat-lim-cs-intros]:
  assumes u : r <_CF.lim 𝔉 : 𝔅 ↦↦_Cα ℭ
    and 𝔊 : 𝔄 ↦↦_C.isoα 𝔅
    and 𝔄′ = 𝔉 ∘_CF 𝔊
  shows u ∘_NTCF-CF 𝔊 : r <_CF.lim 𝔄′ : 𝔄 ↦↦_Cα ℭ
  using assms(1,2)
  unfolding assms(3)
  by (rule ntcf-cf-comp-is-cat-limit-if-is-iso-functor)
```

3.3 Small limit and small colimit

3.3.1 Definition and elementary properties

The concept of a limit is introduced in Chapter III-4 in [9]; the concept of a colimit is introduced in Chapter III-3 in [9]. The definitions of small limits were tailored for ZFC in HOL.

locale *is-tm-cat-limit* = *is-tm-cat-cone* α r $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u **for** α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u +
  assumes *tm-cat-lim-ua-fo*:
    $\bigwedge u'\ r'.\ u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \Longrightarrow$
      $\exists! f'.\ f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \bullet_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f'$

syntax *-is-tm-cat-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} <_{CF.tm.lim} \text{-} :/ \text{-} \mapsto\mapsto_{C.tm1} \text{-})\rangle\ [51, 51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-tm-cat-limit* $\rightleftharpoons$ *is-tm-cat-limit*
translations $u : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-tm-cat-limit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

locale *is-tm-cat-colimit* = *is-tm-cat-cocone* α r $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u **for** α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u +
  assumes *tm-cat-colim-ua-of*:
    $\bigwedge u'\ r'.\ u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \Longrightarrow$
      $\exists! f'.\ f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f' \bullet_{NTCF} u$

syntax *-is-tm-cat-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} >_{CF.tm.colim} \text{-} :/ \text{-} \mapsto\mapsto_{C.tm1} \text{-})\rangle\ [51, 51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-tm-cat-colimit* $\rightleftharpoons$ *is-tm-cat-colimit*
translations $u : \mathfrak{F} >_{CF.tm.colim} r : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-tm-cat-colimit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

Rules.

lemma (**in** *is-tm-cat-limit*) *is-tm-cat-limit-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
  shows $u : r' <_{CF.tm.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-tm-cat-limit-axioms*)

mk-ide rf *is-tm-cat-limit-def*[*unfolded is-tm-cat-limit-axioms-def*]
  |*intro is-tm-cat-limitI*|
  |*dest is-tm-cat-limitD*[*dest*]|
  |*elim is-tm-cat-limitE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-limitD*(*1*)

lemma (**in** *is-tm-cat-colimit*) *is-tm-cat-colimit-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
  shows $u : \mathfrak{F}' >_{CF.tm.colim} r' : \mathfrak{J}' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-tm-cat-colimit-axioms*)

mk-ide rf *is-tm-cat-colimit-def*[*unfolded is-tm-cat-colimit-axioms-def*]
  |*intro is-tm-cat-colimitI*|
  |*dest is-tm-cat-colimitD*[*dest*]|
  |*elim is-tm-cat-colimitE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-colimitD*(*1*)

lemma *is-tm-cat-limitI'*:
  assumes $u : r <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
    and $\bigwedge u'\ r'.\ u' : r' <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \Longrightarrow$
      $\exists! f'.\ f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \bullet_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f'$
  shows $u : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$

proof(*rule is-tm-cat-limitI*, *rule assms*(*1*))
interpret *is-tm-cat-cone* α r $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u **by** (*rule assms*(*1*))
fix r' u' **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
then interpret u': *is-cat-cone* α r' $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u' .
have $u' : r' <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by
(
intro
is-tm-cat-coneI
NTCod.is-tm-functor-axioms
u'.*cat-cone-obj*
u'.*is-ntcf-axioms*
)
then show $\exists! f'.\ f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} \textit{ntcf-const}\ \mathfrak{J}\ \mathfrak{C}\ f'$
by (*rule assms*(*2*))
qed

lemma *is-tm-cat-colimitI'*:
assumes $u : \mathfrak{F} >_{CF.tm.cocone} r : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and $\bigwedge u'\ r'.\ u' : \mathfrak{F} >_{CF.tm.cocone} r' : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \Longrightarrow$
$\exists! f'.\ f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = \textit{ntcf-const}\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF} u$
shows $u : \mathfrak{F} >_{CF.tm.colim} r : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*intro is-tm-cat-colimitI*, *rule assms*(*1*))
interpret *is-tm-cat-cocone* α r $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u **by** (*rule assms*(*1*))
fix r' u' **assume** *prems*: $u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
then interpret u': *is-cat-cocone* α r' $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u' .
have $u' : \mathfrak{F} >_{CF.tm.cocone} r' : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by
(
intro
is-tm-cat-coconeI
NTDom.is-tm-functor-axioms
u'.*cat-cocone-obj*
u'.*is-ntcf-axioms*
)
then show $\exists! f'.\ f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = \textit{ntcf-const}\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF} u$
by (*rule assms*(*2*))
qed

Elementary properties.

sublocale *is-tm-cat-limit* $\subseteq$ *is-cat-limit*
by (*intro is-cat-limitI*, *rule is-cat-cone-axioms*, *rule tm-cat-lim-ua-fo*)

sublocale *is-tm-cat-colimit* $\subseteq$ *is-cat-colimit*
by (*intro is-cat-colimitI*, *rule is-cat-cocone-axioms*, *rule tm-cat-colim-ua-of*)

lemma (**in** *is-cat-limit*) *cat-lim-is-tm-cat-limit*:
assumes $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
shows $u : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*intro is-tm-cat-limitI*)
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*)
show $u : r <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro is-tm-cat-coneI assms is-ntcf-axioms cat-cone-obj*)
fix u' r' **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
show $\exists! f'.\ f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} \textit{ntcf-const}\ \mathfrak{J}\ \mathfrak{C}\ f'$
by (*rule cat-lim-ua-fo*[*OF prems*])
qed

lemma (**in** *is-cat-colimit*) *cat-colim-is-tm-cat-colimit*:
assumes $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
shows $u : \mathfrak{F} >_{CF.tm.colim} r : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*intro is-tm-cat-colimitI*)
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*)
show $u : \mathfrak{F} >_{CF.tm.cocone} r : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro is-tm-cat-coconeI assms is-ntcf-axioms cat-cocone-obj*)
fix u' r' **assume** *prems*: $u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
show $\exists! f'.\ f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f' \bullet_{NTCF} u$
by (*rule cat-colim-ua-of*[*OF prems*])
qed

Limits, colimits and universal arrows.

lemma (**in** *is-tm-cat-limit*) *tm-cat-lim-is-universal-arrow-fo*:
universal-arrow-fo ($\Delta_{CF.tm}$ α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u)
proof(*intro is-functor.universal-arrow-foI*)

show $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Funct* α $\mathfrak{J}$ $\mathfrak{C}$
by
(
intro
tm-cf-diagonal-is-functor
NTCod.HomDom.tiny-category-axioms
NTDom.HomCod.category-axioms
)

show $r \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** (*intro cat-cone-obj*)
then show *ntcf-arrow* $u : \Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{cat\text{-}Funct\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* $\mathfrak{F}$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

fix r' u' **assume** *prems*:
$r' \in_\circ \mathfrak{C}(\!|Obj|\!)$ $u' : \Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|ObjMap|\!)(\!|r'|\!) \mapsto_{cat\text{-}Funct\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* $\mathfrak{F}$
from *prems*(*1*) **have** [*cat-cs-simps*]:
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* $\mathfrak{F}$) = $\mathfrak{F}$
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r')) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r'
by (*cs-concl* **cs-simp**: *cat-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros*)+
from *prems*(*2,1*) **have**
u' : *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r') $\mapsto_{cat\text{-}Funct\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* $\mathfrak{F}$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps*)
note u'[*unfolded cat-cs-simps*] = *cat-Funct-is-arrD*[*OF this*]

from
tm-cat-lim-ua-fo[
OF is-cat-coneI[*OF is-tm-ntcfD*(*1*)[*OF* u'(*1*)] *prems*(*1*)]
]
obtain f
where f: $f : r' \mapsto_{\mathfrak{C}} r$
and [*symmetric*, *cat-cs-simps*]:
ntcf-of-ntcf-arrow $\mathfrak{J}$ $\mathfrak{C}$ u' = $u \bullet_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f
and *f-unique*:
⟦
$f' : r' \mapsto_{\mathfrak{C}} r$;
ntcf-of-ntcf-arrow $\mathfrak{J}$ $\mathfrak{C}$ u' = $u \bullet_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f'

```
      ⟧ ⟹ f' = f
    for f'
    by metis

  show ∃!f'.
    f' : r' ↦_𝔈 r ∧
    u' = umap-fo (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u) r'(|ArrVal|)(|f'|)
  proof(intro ex1I conjI; (elim conjE)?)
    show f : r' ↦_𝔈 r by (rule f)
    with cat-cone-obj show u'-def:
      u' = umap-fo (Δ_CF.tm  α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u) r'(|ArrVal|)(|f|)
      by
        (
          cs-concl
            cs-simp: u'(2)[symmetric] cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    fix f' assume prems':
      f' : r' ↦_𝔈 r
      u' = umap-fo (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u) r'(|ArrVal|)(|f'|)
    from prems'(2) f prems' cat-cone-obj have u'-def':
      u' = ntcf-arrow (u ∘_NTCF ntcf-const 𝔍 𝔈 f')
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from prems'(1) have ntcf-of-ntcf-arrow 𝔍 𝔈 u' = u ∘_NTCF ntcf-const 𝔍 𝔈 f'
      by (cs-concl cs-simp: cat-FUNCT-cs-simps u'-def' cs-intro: cat-cs-intros)
    from f-unique[OF prems'(1) this] show f' = f .

  qed

qed

lemma (in is-tm-cat-cone) tm-cat-cone-is-tm-cat-limit:
  assumes universal-arrow-fo (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑)
  shows 𝔑 : c <_CF.tm.lim 𝔉 : 𝔍 ↦↦_C.tmα 𝔈
proof(intro is-tm-cat-limitI' is-tm-cat-cone-axioms)

  fix u' c' assume prems: u' : c' <_CF.tm.cone 𝔉 : 𝔍 ↦↦_C.tmα 𝔈

  interpret u': is-tm-cat-cone α c' 𝔍 𝔈 𝔉 u' by (rule prems)

  from u'.tm-cat-cone-obj have u'-is-arr:
    ntcf-arrow u' : Δ_CF.tm α 𝔍 𝔈(|ObjMap|)(|c'|) ↦_cat-Funct α 𝔍 𝔈 cf-map 𝔉
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  from is-functor.universal-arrow-foD(3)
    [
      OF
        tm-cf-diagonal-is-functor[
```

```
      OF NTCod.HomDom.tiny-category-axioms NTDom.HomCod.category-axioms
      ]
    assms
    u'.cat-cone-obj
    u'-is-arr
  ]
 obtain f where f: f : c' ↦_𝔈 c
  and u'-def': ntcf-arrow u' =
   umap-fo (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(|ArrVal|)(|f|)
  and f'-unique:
   [[
    f' : c' ↦_𝔈 c;
    ntcf-arrow u' =
     umap-fo (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(|ArrVal|)(|f'|)
   ]] ⟹ f' = f
  for f'
  by metis

 from u'-def' f cat-cone-obj have u'-def: u' = 𝔑 ∘_NTCF ntcf-const 𝔍 𝔈 f
  by
   (
    cs-prems
     cs-simp: cat-cs-simps cat-FUNCT-cs-simps
     cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
   )

 show ∃!f'. f' : c' ↦_𝔈 c ∧ u' = 𝔑 ∘_NTCF ntcf-const 𝔍 𝔈 f'
 proof(intro ex1I conjI; (elim conjE)?, (rule f)?, (rule u'-def)?)
  fix f'' assume prems':
   f'' : c' ↦_𝔈 c u' = 𝔑 ∘_NTCF ntcf-const 𝔍 𝔈 f''
  from prems' have
   ntcf-arrow u' =
    umap-fo (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(|ArrVal|)(|f''|)
   by
    (
     cs-concl
      cs-simp: cat-cs-simps cat-FUNCT-cs-simps
      cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )
  from f'-unique[OF prems'(1) this] show f'' = f.
 qed

qed

lemma (in is-tm-cat-colimit) tm-cat-colim-is-universal-arrow-of:
 universal-arrow-of (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) r (ntcf-arrow u)
proof(intro is-functor.universal-arrow-ofI)

 show Δ_CF.tm α 𝔍 𝔈 : 𝔈 ↦↦_Cα cat-Funct α 𝔍 𝔈
  by
   (
    intro
     tm-cf-diagonal-is-functor
     NTDom.HomDom.tiny-category-axioms
     NTDom.HomCod.category-axioms
   )

 show r ∈₀ 𝔈(|Obj|) by (intro cat-cocone-obj)
```

then show *ntcf-arrow u* : *cf-map* $\mathfrak{F} \mapsto_{\textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}} (\!|\textit{ObjMap}|\!)(\!|r|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

fix r' u' **assume** *prems*:
$r' \in_\circ \mathfrak{C}(\!|\textit{Obj}|\!)$ u' : *cf-map* $\mathfrak{F} \mapsto_{\textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}} (\!|\textit{ObjMap}|\!)(\!|r'|\!)$
from *prems*(*1*) **have** [*cat-cs-simps*]:
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* $\mathfrak{F}$) = $\mathfrak{F}$
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r')) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r'
by (*cs-concl* **cs-simp**: *cat-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros*)+
from *prems*(*2*,*1*) **have**
u' : *cf-map* $\mathfrak{F} \mapsto_{\textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}}$ *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r')
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps*)
note u'[*unfolded cat-cs-simps*] = *cat-Funct-is-arrD*[*OF this*]

from *cat-colim-ua-of*[*OF is-cat-coconeI*[*OF is-tm-ntcfD*(*1*)[*OF* u'(*1*)] *prems*(*1*)]]
obtain f
where f: $f : r \mapsto_{\mathfrak{C}} r'$
and [*symmetric*, *cat-cs-simps*]:
ntcf-of-ntcf-arrow $\mathfrak{J}$ $\mathfrak{C}$ u' = *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f $\cdot_{NTCF}$ u
and *f-unique*:
⟦
$f' : r \mapsto_{\mathfrak{C}} r'$;
ntcf-of-ntcf-arrow $\mathfrak{J}$ $\mathfrak{C}$ u' = *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ $f' \cdot_{NTCF}$ u
⟧ ⟹ f' = f
for f'
by *metis*

show $\exists! f'.$
$f' : r \mapsto_{\mathfrak{C}} r' \wedge$
u' = *umap-of* ($\Delta_{CF.tm}$ α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u) $r'(\!|\textit{ArrVal}|\!)(\!|f'|\!)$
proof(*intro ex1I conjI*; (*elim conjE*)*?*)

show $f : r \mapsto_{\mathfrak{C}} r'$ **by** (*rule f*)

with *cat-cocone-obj* **show** u'-*def*:
u' = *umap-of* ($\Delta_{CF.tm}$ α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u) $r'(\!|\textit{ArrVal}|\!)(\!|f|\!)$
by
(
cs-concl
cs-simp: u'(*2*)[*symmetric*] *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

fix f' **assume** *prems'*:
$f' : r \mapsto_{\mathfrak{C}} r'$
u' = *umap-of* ($\Delta_{CF.tm}$ α $\mathfrak{J}$ $\mathfrak{C}$) (*cf-map* $\mathfrak{F}$) r (*ntcf-arrow* u) $r'(\!|\textit{ArrVal}|\!)(\!|f'|\!)$
from *prems'*(*2*) f *prems'* *cat-cocone-obj* **have** u'-*def'*:
u' = *ntcf-arrow* (*ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ $f' \cdot_{NTCF}$ u)
by
(
cs-prems
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*

```
        cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
    from prems'(1) have ntcf-of-ntcf-arrow 𝔍 𝔈 u' = ntcf-const 𝔍 𝔈 f' ∘_NTCF u
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-FUNCT-cs-simps u'-def' cs-intro: cat-cs-intros
        )
    from f-unique[OF prems'(1) this] show f' = f .

  qed

qed

lemma (in is-tm-cat-cocone) tm-cat-cocone-is-tm-cat-colimit:
  assumes universal-arrow-of (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑)
  shows 𝔑 : 𝔉 >_CF.tm.colim c : 𝔍 ↦↦_C.tmα 𝔈
proof(intro is-tm-cat-colimitI' is-tm-cat-cocone-axioms)

  fix u' c' assume prems: u' : 𝔉 >_CF.tm.cocone c' : 𝔍 ↦↦_C.tmα 𝔈

  interpret u': is-tm-cat-cocone α c' 𝔍 𝔈 𝔉 u' by (rule prems)

  from u'.tm-cat-cocone-obj have u'-is-arr:
    ntcf-arrow u' : cf-map 𝔉 ↦_cat-Funct α 𝔍 𝔈 Δ_CF.tm α 𝔍 𝔈(⦇ObjMap⦈)(⦇c'⦈)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  from is-functor.universal-arrow-ofD(3)
    [
      OF
        tm-cf-diagonal-is-functor[
          OF NTDom.HomDom.tiny-category-axioms NTDom.HomCod.category-axioms
          ]
        assms
        u'.cat-cocone-obj
        u'-is-arr
    ]
  obtain f where f: f : c ↦_𝔈 c'
    and u'-def': ntcf-arrow u' =
      umap-of (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(⦇ArrVal⦈)(⦇f⦈)
    and f'-unique:
      ⟦
        f' : c ↦_𝔈 c';
        ntcf-arrow u' =
          umap-of (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(⦇ArrVal⦈)(⦇f'⦈)
      ⟧ ⟹ f' = f
    for f'
    by metis

  from u'-def' f cat-cocone-obj have u'-def: u' = ntcf-const 𝔍 𝔈 f ∘_NTCF 𝔑
    by
      (
        cs-prems
```

```
        cs-simp: cat-cs-simps cat-FUNCT-cs-simps
        cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )

  show ∃!f'. f' : c ↦_𝔈 c' ∧ u' = ntcf-const 𝔍 𝔈 f' ∙_NTCF 𝔑
  proof(intro ex1I conjI; (elim conjE)?, (rule f)?, (rule u'-def)?)
    fix f'' assume prems':
      f'' : c ↦_𝔈 c' u' = ntcf-const 𝔍 𝔈 f'' ∙_NTCF 𝔑
    from prems' have
      ntcf-arrow u' =
        umap-of (Δ_CF.tm α 𝔍 𝔈) (cf-map 𝔉) c (ntcf-arrow 𝔑) c'(|ArrVal|)(|f''|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from f'-unique[OF prems'(1) this] show f'' = f.
  qed

qed
```

Duality.

```
lemma (in is-tm-cat-limit) is-tm-cat-colimit-op:
  op-ntcf u : op-cf 𝔉 >_CF.tm.colim r : op-cat 𝔍 ↦↦_C.tmα op-cat 𝔈
proof(intro is-tm-cat-colimitI')
  show op-ntcf u : op-cf 𝔉 >_CF.tm.cocone r : op-cat 𝔍 ↦↦_C.tmα op-cat 𝔈
    by (cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros)
  fix u' r' assume prems:
    u' : op-cf 𝔉 >_CF.tm.cocone r' : op-cat 𝔍 ↦↦_C.tmα op-cat 𝔈
  interpret u': is-tm-cat-cocone α r' ‹op-cat 𝔍› ‹op-cat 𝔈› ‹op-cf 𝔉› u'
    by (rule prems)
  from tm-cat-lim-ua-fo[OF u'.is-cat-cone-op[unfolded cat-op-simps]] obtain f
    where f: f : r' ↦_𝔈 r
      and op-u'-def: op-ntcf u' = u ∙_NTCF ntcf-const 𝔍 𝔈 f
      and f-unique:
        ⟦ f' : r' ↦_𝔈 r; op-ntcf u' = u ∙_NTCF ntcf-const 𝔍 𝔈 f' ⟧ ⟹
          f' = f
    for f'
    by metis
  from op-u'-def have op-ntcf (op-ntcf u') = op-ntcf (u ∙_NTCF ntcf-const 𝔍 𝔈 f)
    by simp
  from this f have u'-def:
    u' = ntcf-const (op-cat 𝔍) (op-cat 𝔈) f ∙_NTCF op-ntcf u
    by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros)
  show ∃!f'.
    f' : r ↦_op-cat 𝔈 r' ∧
    u' = ntcf-const (op-cat 𝔍) (op-cat 𝔈) f' ∙_NTCF op-ntcf u
  proof(intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?)
    fix f' assume prems':
      f' : r' ↦_𝔈 r
      u' = ntcf-const (op-cat 𝔍) (op-cat 𝔈) f' ∙_NTCF op-ntcf u
    from prems'(2) have op-ntcf u' =
      op-ntcf (ntcf-const (op-cat 𝔍) (op-cat 𝔈) f' ∙_NTCF op-ntcf u)
      by simp
    from this prems'(1) have op-ntcf u' = u ∙_NTCF ntcf-const 𝔍 𝔈 f'
      by
        (
```

```
      cs-prems
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro: cat-cs-intros cat-op-intros
    )
  from f-unique[OF prems'(1) this] show f' = f.
 qed (intro u'-def f)+
qed

lemma (in is-tm-cat-limit) is-tm-cat-colimit-op'[cat-op-intros]:
 assumes 𝔉' = op-cf 𝔉 and 𝔍' = op-cat 𝔍 and ℭ' = op-cat ℭ
 shows op-ntcf u : 𝔉' >_{CF.tm.colim} r : 𝔍' ↦↦_{C.tm α} ℭ'
 unfolding assms by (rule is-tm-cat-colimit-op)

lemmas [cat-op-intros] = is-tm-cat-limit.is-tm-cat-colimit-op'

lemma (in is-tm-cat-colimit) is-tm-cat-limit-op:
 op-ntcf u : r <_{CF.tm.lim} op-cf 𝔉 : op-cat 𝔍 ↦↦_{C.tm α} op-cat ℭ
proof(intro is-tm-cat-limitI')
 show op-ntcf u : r <_{CF.tm.cone} op-cf 𝔉 : op-cat 𝔍 ↦↦_{C.tm α} op-cat ℭ
  by (cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros)
 fix u' r' assume prems:
  u' : r' <_{CF.tm.cone} op-cf 𝔉 : op-cat 𝔍 ↦↦_{C.tm α} op-cat ℭ
 interpret u': is-tm-cat-cone α r' ‹op-cat 𝔍› ‹op-cat ℭ› ‹op-cf 𝔉› u'
  by (rule prems)
 from tm-cat-colim-ua-of[OF u'.is-cat-cocone-op[unfolded cat-op-simps]] obtain f
  where f: f : r ↦_ℭ r'
    and op-u'-def: op-ntcf u' = ntcf-const 𝔍 ℭ f ∙_{NTCF} u
    and f-unique:
      ⟦ f' : r ↦_ℭ r'; op-ntcf u' = ntcf-const 𝔍 ℭ f' ∙_{NTCF} u ⟧ ⟹ f' = f
  for f'
  by metis
 from op-u'-def have op-ntcf (op-ntcf u') = op-ntcf (ntcf-const 𝔍 ℭ f ∙_{NTCF} u)
  by simp
 from this f have u'-def:
  u' = op-ntcf u ∙_{NTCF} ntcf-const (op-cat 𝔍) (op-cat ℭ) f
  by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros)
 show ∃!f'.
  f' : r' ↦_{op-cat ℭ} r' ∧
  u' = op-ntcf u ∙_{NTCF} ntcf-const (op-cat 𝔍) (op-cat ℭ) f'
 proof(intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?)
  fix f' assume prems':
   f' : r ↦_ℭ r'
   u' = op-ntcf u ∙_{NTCF} ntcf-const (op-cat 𝔍) (op-cat ℭ) f'
  from prems'(2) have op-ntcf u' =
   op-ntcf (op-ntcf u ∙_{NTCF} ntcf-const (op-cat 𝔍) (op-cat ℭ) f')
   by simp
  from this prems'(1) have op-ntcf u' = ntcf-const 𝔍 ℭ f' ∙_{NTCF} u
   by
    (
     cs-prems
       cs-simp: cat-cs-simps cat-op-simps
       cs-intro: cat-cs-intros cat-op-intros
    )
  from f-unique[OF prems'(1) this] show f' = f.
 qed (intro u'-def f)+
qed

lemma (in is-tm-cat-colimit) is-tm-cat-colimit-op'[cat-op-intros]:
```

assumes $\mathfrak{F}' = \mathit{op\text{-}cf}\ \mathfrak{F}$ **and** $\mathfrak{J}' = \mathit{op\text{-}cat}\ \mathfrak{J}$ **and** $\mathfrak{C}' = \mathit{op\text{-}cat}\ \mathfrak{C}$
shows $\mathit{op\text{-}ntcf}\ u : r <_{CF.tm.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-limit-op*)

lemmas [*cat-op-intros*] = *is-tm-cat-colimit.is-tm-cat-colimit-op'*

3.3.2 Further properties

lemma (**in** *is-tm-cat-limit*) *tm-cat-lim-is-tm-cat-limit-if-iso-arr*:
assumes $f : r' \mapsto_{iso_{\mathfrak{C}}} r$
shows $u \cdot_{NTCF} \mathit{ntcf\text{-}const}\ \mathfrak{J}\ \mathfrak{C}\ f : r' <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof–
note $f = \mathit{is\text{-}iso\text{-}arrD}(1)[\mathit{OF}\ \mathit{assms}]$
from $f(1)$ **interpret** u': $\mathit{is\text{-}tm\text{-}cat\text{-}cone}\ \alpha\ r'\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ \langle u \cdot_{NTCF} \mathit{ntcf\text{-}const}\ \mathfrak{J}\ \mathfrak{C}\ f\rangle$
by (*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros*)
show *?thesis*
proof
(
intro u'.tm-cat-cone-is-tm-cat-limit,
rule is-functor.universal-arrow-fo-if-universal-arrow-fo,
rule tm-cf-diagonal-is-functor,
rule NTCod.HomDom.tiny-category-axioms,
rule NTDom.HomCod.category-axioms,
rule tm-cat-lim-is-universal-arrow-fo
)
show $f : r' \mapsto_{iso_{\mathfrak{C}}} r$ **by** (*rule assms*)
from *f* **show** $\mathit{ntcf\text{-}arrow}\ (u \cdot_{NTCF} \mathit{ntcf\text{-}const}\ \mathfrak{J}\ \mathfrak{C}\ f) =$
$\mathit{umap\text{-}fo}\ (\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C})\ (\mathit{cf\text{-}map}\ \mathfrak{F})\ r\ (\mathit{ntcf\text{-}arrow}\ u)\ r'(\!|\mathit{ArrVal}|\!)(\!|f|\!)$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
qed
qed

lemma (**in** *is-tm-cat-colimit*) *tm-cat-colim-is-tm-cat-colimit-if-iso-arr*:
assumes $f : r \mapsto_{iso_{\mathfrak{C}}} r'$
shows $\mathit{ntcf\text{-}const}\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF} u : \mathfrak{F} >_{CF.tm.colim} r' : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof–
note $f = \mathit{is\text{-}iso\text{-}arrD}(1)[\mathit{OF}\ \mathit{assms}]$
from $f(1)$ **interpret** u':
$\mathit{is\text{-}tm\text{-}cat\text{-}cocone}\ \alpha\ r'\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ \langle \mathit{ntcf\text{-}const}\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF} u\rangle$
by (*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros*)
from *f* **have** [*symmetric*, *cat-op-simps*]:
$\mathit{op\text{-}ntcf}\ (\mathit{ntcf\text{-}const}\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF} u) =$
$\mathit{op\text{-}ntcf}\ u \cdot_{NTCF} \mathit{ntcf\text{-}const}\ (\mathit{op\text{-}cat}\ \mathfrak{J})\ (\mathit{op\text{-}cat}\ \mathfrak{C})\ f$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)
show *?thesis*
by
(
rule is-tm-cat-limit.is-tm-cat-colimit-op
[

OF is-tm-cat-limit.tm-cat-lim-is-tm-cat-limit-if-iso-arr[
OF is-tm-cat-limit-op, *unfolded cat-op-simps*, *OF assms*(*1*)
],
unfolded cat-op-simps
]
)
qed

3.4 Finite limit and finite colimit

locale *is-cat-finite-limit* =
is-cat-limit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u + *NTDom.HomDom*: *finite-category* α $\mathfrak{J}$
for α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

syntax *-is-cat-finite-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $<_{CF.lim.fin}$ - :/ - $\mapsto\mapsto_{C^1}$ -)$\rangle$ [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-finite-limit* $\rightleftharpoons$ *is-cat-finite-limit*
translations $u : r <_{CF.lim.fin} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
CONST is-cat-finite-limit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

locale *is-cat-finite-colimit* =
is-cat-colimit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u + *NTDom.HomDom*: *finite-category* α $\mathfrak{J}$
for α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

syntax *-is-cat-finite-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $>_{CF.colim.fin}$ - :/ - $\mapsto\mapsto_{C^1}$ -)$\rangle$ [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-finite-colimit* $\rightleftharpoons$ *is-cat-finite-colimit*
translations $u : \mathfrak{F} >_{CF.colim.fin} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
CONST is-cat-finite-colimit α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r u

Rules.

lemma (**in** *is-cat-finite-limit*) *is-cat-finite-limit-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : r' <_{CF.lim.fin} \mathfrak{F}' : \mathfrak{J}' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-limit-axioms*)

mk-ide rf *is-cat-finite-limit-def*
|*intro is-cat-finite-limitI*|
|*dest is-cat-finite-limitD*[*dest*]|
|*elim is-cat-finite-limitE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-limitD*

lemma (**in** *is-cat-finite-colimit*)
is-cat-finite-colimit-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : \mathfrak{F}' >_{CF.colim.fin} r' : \mathfrak{J}' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-colimit-axioms*)

mk-ide rf *is-cat-finite-colimit-def*[*unfolded is-cat-colimit-axioms-def*]
|*intro is-cat-finite-colimitI*|
|*dest is-cat-finite-colimitD*[*dest*]|
|*elim is-cat-finite-colimitE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-colimitD*

Duality.

lemma (**in** *is-cat-finite-limit*) *is-cat-finite-colimit-op*:

op-ntcf u : *op-cf* $\mathfrak{F}$ $>_{CF.colim.fin}$ r : *op-cat* $\mathfrak{J}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by
(
cs-concl **cs-shallow**
cs-intro: *is-cat-finite-colimitI cat-op-intros cat-small-cs-intros*
)

lemma (**in** *is-cat-finite-limit*) *is-cat-finite-colimit-op'*[*cat-op-intros*]:
assumes $\mathfrak{F}'$ = *op-cf* $\mathfrak{F}$ **and** $\mathfrak{J}'$ = *op-cat* $\mathfrak{J}$ **and** $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* u : $\mathfrak{F}'$ $>_{CF.colim.fin}$ r : $\mathfrak{J}'$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-colimit-op*)

lemmas [*cat-op-intros*] = *is-cat-finite-limit.is-cat-finite-colimit-op'*

lemma (**in** *is-cat-finite-colimit*) *is-cat-finite-limit-op*:
op-ntcf u : r $<_{CF.lim.fin}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{J}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by
(
cs-concl **cs-shallow**
cs-intro: *is-cat-finite-limitI cat-op-intros cat-small-cs-intros*
)

lemma (**in** *is-cat-finite-colimit*) *is-cat-finite-colimit-op'*[*cat-op-intros*]:
assumes $\mathfrak{F}'$ = *op-cf* $\mathfrak{F}$ **and** $\mathfrak{J}'$ = *op-cat* $\mathfrak{J}$ **and** $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* u : r $<_{CF.lim.fin}$ $\mathfrak{F}'$: $\mathfrak{J}'$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-limit-op*)

lemmas [*cat-op-intros*] = *is-cat-finite-colimit.is-cat-finite-colimit-op'*

Elementary properties.

sublocale *is-cat-finite-limit* ⊆ *is-tm-cat-limit*
by
(
intro
is-tm-cat-limitI
is-tm-cat-coneI
is-ntcf-axioms
cat-lim-ua-fo
cat-cone-obj
NTCod.cf-is-tm-functor-if-HomDom-finite-category[
OF NTDom.HomDom.finite-category-axioms
]
)

sublocale *is-cat-finite-colimit* ⊆ *is-tm-cat-colimit*
by
(
intro
is-tm-cat-colimitI
is-tm-cat-coconeI
is-ntcf-axioms
cat-colim-ua-of
cat-cocone-obj
NTDom.cf-is-tm-functor-if-HomDom-finite-category[
OF NTDom.HomDom.finite-category-axioms
]
)

3.5 Creation of limits

See Chapter V-1 in [9].

definition *cf-creates-limits* :: $V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *cf-creates-limits* α $\mathfrak{G}$ $\mathfrak{F}$ =
  (
    $\forall \tau\ b.$
      $\tau : b <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{F}(\!|HomDom|\!) \mapsto\mapsto_{C\alpha} \mathfrak{G}(\!|HomCod|\!) \longrightarrow$
      (
        (
          $\exists !\sigma a.\ \exists \sigma\ a.\ \sigma a = \langle \sigma,\ a \rangle \wedge$
            $\sigma : a <_{CF.cone} \mathfrak{F} : \mathfrak{F}(\!|HomDom|\!) \mapsto\mapsto_{C\alpha} \mathfrak{F}(\!|HomCod|\!) \wedge$
            $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma \wedge$
            $b = \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$
) $\wedge$
        (
          $\forall \sigma\ a.$
            $\sigma : a <_{CF.cone} \mathfrak{F} : \mathfrak{F}(\!|HomDom|\!) \mapsto\mapsto_{C\alpha} \mathfrak{F}(\!|HomCod|\!) \longrightarrow$
            $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma \longrightarrow$
            $b = \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \longrightarrow$
            $\sigma : a <_{CF.lim} \mathfrak{F} : \mathfrak{F}(\!|HomDom|\!) \mapsto\mapsto_{C\alpha} \mathfrak{F}(\!|HomCod|\!)$
)
)
)

Rules.

context
  fixes α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{F}$
  assumes $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
begin

interpretation $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
interpretation $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)

mk-ide rf *cf-creates-limits-def*[
  where α=α **and** $\mathfrak{F}$=$\mathfrak{F}$ **and** $\mathfrak{G}$=$\mathfrak{G}$, *unfolded cat-cs-simps*
  ]
  |*intro cf-creates-limitsI*|
  |*dest cf-creates-limitsD'*|
  |*elim cf-creates-limitsE'*|

end

lemmas *cf-creates-limitsD*[*dest*!] = *cf-creates-limitsD'*[*rotated 2*]
  and *cf-creates-limitsE*[*elim*!] = *cf-creates-limitsE'*[*rotated 2*]

lemma *cf-creates-limitsE''*:
  assumes *cf-creates-limits* α $\mathfrak{G}$ $\mathfrak{F}$
    and $\tau : b <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
  obtains σ r **where** $\sigma : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma$
    and $b = \mathfrak{G}(\!|ObjMap|\!)(\!|r|\!)$
proof–
  note *cfID* = *cf-creates-limitsD*[*OF assms*]
  from *conjunct1*[*OF cfID*] **obtain** σ r

```
    where σ: σ : r <_{CF.cone} ℱ : 𝔍 ↦↦_{Cα} 𝔄
      and τ-def: τ = 𝔊 ∘_{CF-NTCF} σ
      and b-def: b = 𝔊(⁅ObjMap⁆)(⁅r⁆)
    by metis
  moreover have σ : r <_{CF.lim} ℱ : 𝔍 ↦↦_{Cα} 𝔄
    by (rule conjunct2[OF cflD, rule-format, OF σ τ-def b-def])
  ultimately show ?thesis using that by auto
qed
```

3.6 Preservation of limits and colimits

3.6.1 Definitions and elementary properties

See Chapter V-4 in [9].

```
definition cf-preserves-limits :: V ⇒ V ⇒ V ⇒ bool
  where cf-preserves-limits α 𝔊 ℱ =
    (
      ∀σ a.
        σ : a <_{CF.lim} ℱ : ℱ(⁅HomDom⁆) ↦↦_{Cα} ℱ(⁅HomCod⁆) ⟶
        𝔊 ∘_{CF-NTCF} σ : 𝔊(⁅ObjMap⁆)(⁅a⁆) <_{CF.lim} 𝔊 ∘_{CF} ℱ : ℱ(⁅HomDom⁆) ↦↦_{Cα} 𝔊(⁅HomCod⁆)
    )

definition cf-preserves-colimits :: V ⇒ V ⇒ V ⇒ bool
  where cf-preserves-colimits α 𝔊 ℱ =
    (
      ∀σ a.
        σ : ℱ >_{CF.colim} a : ℱ(⁅HomDom⁆) ↦↦_{Cα} ℱ(⁅HomCod⁆) ⟶
        𝔊 ∘_{CF-NTCF} σ : 𝔊 ∘_{CF} ℱ >_{CF.colim} 𝔊(⁅ObjMap⁆)(⁅a⁆) : ℱ(⁅HomDom⁆) ↦↦_{Cα} 𝔊(⁅HomCod⁆)
    )
```

Rules.

```
context
  fixes α 𝔍 𝔄 𝔅 𝔊 ℱ
  assumes ℱ: ℱ : 𝔍 ↦↦_{Cα} 𝔄
    and 𝔊: 𝔊 : 𝔄 ↦↦_{Cα} 𝔅
begin

interpretation ℱ: is-functor α 𝔍 𝔄 ℱ by (rule ℱ)
interpretation 𝔊: is-functor α 𝔄 𝔅 𝔊 by (rule 𝔊)

mk-ide rf cf-preserves-limits-def[
  where α=α and ℱ=ℱ and 𝔊=𝔊, unfolded cat-cs-simps
  ]
  |intro cf-preserves-limitsI|
  |dest cf-preserves-limitsD'|
  |elim cf-preserves-limitsE'|

mk-ide rf cf-preserves-colimits-def[
  where α=α and ℱ=ℱ and 𝔊=𝔊, unfolded cat-cs-simps
  ]
  |intro cf-preserves-colimitsI|
  |dest cf-preserves-colimitsD'|
  |elim cf-preserves-colimitsE'|

end

lemmas cf-preserves-limitsD[dest!] = cf-preserves-limitsD'[rotated 2]
  and cf-preserves-limitsE[elim!] = cf-preserves-limitsE'[rotated 2]
```

lemmas *cf-preserves-colimitsD*[*dest*!] = *cf-preserves-colimitsD'*[*rotated 2*]
and *cf-preserves-colimitsE*[*elim*!] = *cf-preserves-colimitsE'*[*rotated 2*]

Duality.

lemma *cf-preserves-colimits-op*[*cat-op-simps*]:
assumes $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows
cf-preserves-colimits α (*op-cf* $\mathfrak{G}$) (*op-cf* $\mathfrak{F}$) $\longleftrightarrow$
cf-preserves-limits α $\mathfrak{G}$ $\mathfrak{F}$
proof

interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))

show *cf-preserves-limits* α $\mathfrak{G}$ $\mathfrak{F}$
if *cf-preserves-colimits* α (*op-cf* $\mathfrak{G}$) (*op-cf* $\mathfrak{F}$)
proof(*rule cf-preserves-limitsI*, *rule assms*(*1*), *rule assms*(*2*))
fix σ r **assume** $\sigma : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
then interpret σ: *is-cat-limit* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ r σ .
show $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor) <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by
(
rule is-cat-colimit.is-cat-limit-op
[
OF cf-preserves-colimitsD
[
OF that σ.*is-cat-colimit-op* $\mathfrak{F}$.*is-functor-op* $\mathfrak{G}$.*is-functor-op*,
folded op-cf-cf-comp op-ntcf-cf-ntcf-comp
],
unfolded cat-op-simps
]
)
qed

show *cf-preserves-colimits* α (*op-cf* $\mathfrak{G}$) (*op-cf* $\mathfrak{F}$)
if *cf-preserves-limits* α $\mathfrak{G}$ $\mathfrak{F}$
proof
(
rule cf-preserves-colimitsI,
rule $\mathfrak{F}$.*is-functor-op*,
rule $\mathfrak{G}$.*is-functor-op*,
unfold cat-op-simps
)
fix σ r **assume** σ : *op-cf* $\mathfrak{F}$ $>_{CF.colim}$ r : *op-cat* $\mathfrak{J}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{A}$
then interpret σ: *is-cat-colimit* α ‹*op-cat* $\mathfrak{J}$› ‹*op-cat* $\mathfrak{A}$› ‹*op-cf* $\mathfrak{F}$› r σ .
show *op-cf* $\mathfrak{G}$ $\circ_{CF-NTCF}$ σ :
op-cf $\mathfrak{G}$ $\circ_{CF}$ *op-cf* $\mathfrak{F}$ $>_{CF.colim}$ $\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$: *op-cat* $\mathfrak{J}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{B}$
by
(
rule is-cat-limit.is-cat-colimit-op
[
OF cf-preserves-limitsD[
OF that σ.*is-cat-limit-op*[*unfolded cat-op-simps*] *assms*(*1*,*2*)
],
unfolded cat-op-simps
]
)

```
  qed

qed

lemma cf-preserves-limits-op[cat-op-simps]:
  assumes 𝔉 : 𝔍 ↦↦_{Cα} 𝔄 and 𝔊 : 𝔄 ↦↦_{Cα} 𝔅
  shows
    cf-preserves-limits α (op-cf 𝔊) (op-cf 𝔉) ⟷
      cf-preserves-colimits α 𝔊 𝔉
proof

  interpret 𝔉: is-functor α 𝔍 𝔄 𝔉 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔄 𝔅 𝔊 by (rule assms(2))

  show cf-preserves-colimits α 𝔊 𝔉
    if cf-preserves-limits α (op-cf 𝔊) (op-cf 𝔉)
  proof(rule cf-preserves-colimitsI, rule assms(1), rule assms(2))
    fix σ r assume σ : 𝔉 >_{CF.colim} r : 𝔍 ↦↦_{Cα} 𝔄
    then interpret σ: is-cat-colimit α 𝔍 𝔄 𝔉 r σ .
    show 𝔊 ∘_{CF-NTCF} σ : 𝔊 ∘_{CF} 𝔉 >_{CF.colim} 𝔊(|ObjMap|)(|r|) : 𝔍 ↦↦_{Cα} 𝔅
      by
        (
          rule is-cat-limit.is-cat-colimit-op
            [
              OF cf-preserves-limitsD
                [
                  OF that σ.is-cat-limit-op 𝔉.is-functor-op 𝔊.is-functor-op,
                  folded op-cf-cf-comp op-ntcf-cf-ntcf-comp
                ],
              unfolded cat-op-simps
            ]
        )
  qed

  show cf-preserves-limits α (op-cf 𝔊) (op-cf 𝔉)
    if cf-preserves-colimits α 𝔊 𝔉
  proof
    (
      rule cf-preserves-limitsI,
      rule 𝔉.is-functor-op,
      rule 𝔊.is-functor-op,
      unfold cat-op-simps
    )
    fix σ r assume σ : r <_{CF.lim} op-cf 𝔉 : op-cat 𝔍 ↦↦_{Cα} op-cat 𝔄
    then interpret σ: is-cat-limit α ‹op-cat 𝔍› ‹op-cat 𝔄› ‹op-cf 𝔉› r σ .
    show op-cf 𝔊 ∘_{CF-NTCF} σ :
      𝔊(|ObjMap|)(|r|) <_{CF.lim} op-cf 𝔊 ∘_{CF} op-cf 𝔉 : op-cat 𝔍 ↦↦_{Cα} op-cat 𝔅
      by
        (
          rule is-cat-colimit.is-cat-limit-op
            [
              OF cf-preserves-colimitsD[
                OF that σ.is-cat-colimit-op[unfolded cat-op-simps] assms(1,2)
                ],
              unfolded cat-op-simps
            ]
        )
  qed
```

qed

3.6.2 Further properties

lemma *cf-preserves-limits-if-cf-creates-limits*:
— See Theorem 2 in Chapter V-4 in [9].
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $\psi : b <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *cf-creates-limits* α $\mathfrak{G}$ $\mathfrak{F}$
shows *cf-preserves-limits* α $\mathfrak{G}$ $\mathfrak{F}$
proof–

interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
interpret ψ: *is-cat-limit* α $\mathfrak{J}$ $\mathfrak{B}$ $\langle\mathfrak{G} \circ_{CF} \mathfrak{F}\rangle$ b ψ
by (*intro is-cat-limit.cat-lim-is-tm-cat-limit assms*(*3*,*4*))

show *?thesis*
proof
(
intro cf-preserves-limitsI,
rule $\mathfrak{F}$*.is-functor-axioms*,
rule $\mathfrak{G}$*.is-functor-axioms*
)

fix σ a **assume** *prems*: $\sigma : a <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
then interpret σ: *is-cat-limit* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ a σ **.**

obtain τ A
where τ: $\tau : A <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and ψ*-def*: $\psi = \mathfrak{G} \circ_{CF-NTCF} \tau$
and *b-def*: $b = \mathfrak{G}(\!|ObjMap|\!)(\!|A|\!)$
by
(
rule cf-creates-limitsE''
[
OF
assms(*4*)
ψ*.is-cat-limit-axioms*
$\mathfrak{F}$*.is-functor-axioms*
$\mathfrak{G}$*.is-functor-axioms*
]
)
from τ **interpret** τ: *is-cat-limit* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ A τ **.**

from *cat-lim-ex-is-iso-arr*[*OF* τ*.is-cat-limit-axioms prems*] **obtain** f
where f: $f : a \mapsto_{iso\mathfrak{A}} A$ **and** σ*-def*: $\sigma = \tau \cdot_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{A}$ f
by *auto*

note $f = f$ *is-iso-arrD*(*1*)[*OF* f]

from f(*2*) **have** $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) <_{CF.cone} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*intro is-cat-coneI*)
(*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

from σ*-def* **have** $\mathfrak{G} \circ_{CF-NTCF} \sigma = \mathfrak{G} \circ_{CF-NTCF} (\tau \cdot_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{A}$ $f)$

by *simp*
also from *f(2)* **have** ... = ψ $\cdot_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{G}(\!|ArrMap|\!)(\!|f|\!)$)
by (*cs-concl-step cf-ntcf-comp-ntcf-vcomp*)
(
cs-concl
cs-simp: *cat-cs-simps* ψ*-def*[*symmetric*] **cs-intro**: *cat-cs-intros*
)
finally have $\mathfrak{G}\sigma$: $\mathfrak{G}$ $\circ_{CF-NTCF}$ σ = ψ $\cdot_{NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{G}(\!|ArrMap|\!)(\!|f|\!)$) .

show $\mathfrak{G}$ $\circ_{CF-NTCF}$ σ : $\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$ $<_{CF.lim}$ $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
by
(
rule ψ*.cat-lim-is-cat-limit-if-is-iso-arr*
[
OF $\mathfrak{G}$*.cf-ArrMap-is-iso-arr*[*OF f(1)*, *folded b-def*],
folded $\mathfrak{G}\sigma$
]
)

qed

qed

3.7 Continuous and cocontinuous functor

3.7.1 Definition and elementary properties

definition *is-cf-continuous* :: *V* ⇒ *V* ⇒ *bool*
where *is-cf-continuous* α $\mathfrak{G}$ ⟷
($\forall$ $\mathfrak{F}$ $\mathfrak{J}$. $\mathfrak{F}$: $\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{G}(\!|HomDom|\!)$ ⟶ *cf-preserves-limits* α $\mathfrak{G}$ $\mathfrak{F}$)

definition *is-cf-cocontinuous* :: *V* ⇒ *V* ⇒ *bool*
where *is-cf-cocontinuous* α $\mathfrak{G}$ ⟷
($\forall$ $\mathfrak{F}$ $\mathfrak{J}$. $\mathfrak{F}$: $\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{G}(\!|HomDom|\!)$ ⟶ *cf-preserves-colimits* α $\mathfrak{G}$ $\mathfrak{F}$)

Rules.

context
fixes α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{F}$
assumes $\mathfrak{G}$: $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
begin

interpretation $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)

mk-ide rf *is-cf-continuous-def*[**where** α=α **and** $\mathfrak{G}$=$\mathfrak{G}$, *unfolded cat-cs-simps*]
|*intro is-cf-continuousI*|
|*dest is-cf-continuousD′*|
|*elim is-cf-continuousE′*|

mk-ide rf *is-cf-cocontinuous-def*[**where** α=α **and** $\mathfrak{G}$=$\mathfrak{G}$, *unfolded cat-cs-simps*]
|*intro is-cf-cocontinuousI*|
|*dest is-cf-cocontinuousD′*|
|*elim is-cf-cocontinuousE′*|

end

lemmas *is-cf-continuousD*[*dest*!] = *is-cf-continuousD′*[*rotated*]
and *is-cf-continuousE*[*elim*!] = *is-cf-continuousE′*[*rotated*]

lemmas *is-cf-cocontinuousD*[*dest*!] = *is-cf-cocontinuousD′*[*rotated*]
and *is-cf-cocontinuousE*[*elim*!] = *is-cf-cocontinuousE′*[*rotated*]

Duality.

lemma *is-cf-continuous-op*[*cat-op-simps*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *is-cf-continuous* α (*op-cf* $\mathfrak{G}$) $\longleftrightarrow$ *is-cf-cocontinuous* α $\mathfrak{G}$
proof
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
show *is-cf-cocontinuous* α $\mathfrak{G}$ **if** *is-cf-continuous* α (*op-cf* $\mathfrak{G}$)
proof(*intro is-cf-cocontinuousI*, *rule assms*)
fix $\mathfrak{F}$ $\mathfrak{J}$ **assume** *prems′*: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
then interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}$ **.**
show *cf-preserves-colimits* α $\mathfrak{G}$ $\mathfrak{F}$
by
(
rule cf-preserves-limits-op
[
THEN iffD1,
OF
prems′
assms(*1*)
is-cf-continuousD[*OF that* $\mathfrak{F}$.*is-functor-op* $\mathfrak{G}$.*is-functor-op*]
]
)
qed
show *is-cf-continuous* α (*op-cf* $\mathfrak{G}$) **if** *is-cf-cocontinuous* α $\mathfrak{G}$
proof(*intro is-cf-continuousI*, *rule* $\mathfrak{G}$.*is-functor-op*)
fix $\mathfrak{F}$ $\mathfrak{J}$ **assume** *prems′*: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{A}$
then interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ ‹*op-cat* $\mathfrak{A}$› $\mathfrak{F}$ **.**
from *that assms* **have** *op-op-bundle*:
is-cf-cocontinuous α (*op-cf* (*op-cf* $\mathfrak{G}$))
op-cf (*op-cf* $\mathfrak{G}$) : $\mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
unfolding *cat-op-simps* **.**
show *cf-preserves-limits* α (*op-cf* $\mathfrak{G}$) $\mathfrak{F}$
by
(
rule cf-preserves-colimits-op
[
THEN iffD1,
OF
$\mathfrak{F}$.*is-functor-axioms*
$\mathfrak{G}$.*is-functor-op*
is-cf-cocontinuousD
[
OF
op-op-bundle(*1*)
$\mathfrak{F}$.*is-functor-op*[*unfolded cat-op-simps*]
op-op-bundle(*2*)
]
]
)
qed
qed

lemma *is-cf-cocontinuous-op*[*cat-op-simps*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *is-cf-cocontinuous* α (*op-cf* $\mathfrak{G}$) $\longleftrightarrow$ *is-cf-continuous* α $\mathfrak{G}$

```
proof
  interpret 𝔊: is-functor α 𝔄 𝔅 𝔊 by (rule assms(1))
  show is-cf-continuous α 𝔊 if is-cf-cocontinuous α (op-cf 𝔊)
  proof(intro is-cf-continuousI, rule assms)
    fix 𝔉 𝔍 assume prems': 𝔉 : 𝔍 ↦↦_Cα 𝔄
    then interpret 𝔉: is-functor α 𝔍 𝔄 𝔉 .
    show cf-preserves-limits α 𝔊 𝔉
      by
        (
          rule cf-preserves-colimits-op
            [
              THEN iffD1,
              OF
                prems'
                assms(1)
                is-cf-cocontinuousD[OF that 𝔉.is-functor-op 𝔊.is-functor-op]
            ]
        )
  qed
  show is-cf-cocontinuous α (op-cf 𝔊) if is-cf-continuous α 𝔊
  proof(intro is-cf-cocontinuousI, rule 𝔊.is-functor-op)
    fix 𝔉 𝔍 assume prems': 𝔉 : 𝔍 ↦↦_Cα op-cat 𝔄
    then interpret 𝔉: is-functor α 𝔍 ‹op-cat 𝔄› 𝔉 .
    from that assms have op-op-bundle:
      is-cf-continuous α (op-cf (op-cf 𝔊))
      op-cf (op-cf 𝔊) : 𝔄 ↦↦_Cα 𝔅
      unfolding cat-op-simps .
    show cf-preserves-colimits α (op-cf 𝔊) 𝔉
      by
        (
          rule cf-preserves-limits-op
            [
              THEN iffD1,
              OF
                𝔉.is-functor-axioms
                𝔊.is-functor-op
                is-cf-continuousD
                  [
                    OF
                      op-op-bundle(1)
                      𝔉.is-functor-op[unfolded cat-op-simps]
                      op-op-bundle(2)
                  ]
            ]
        )
  qed
qed
```

3.7.2 Category isomorphisms are continuous and cocontinuous

```
lemma (in is-iso-functor) iso-cf-is-cf-continuous: is-cf-continuous α 𝔉
proof(intro is-cf-continuousI)
  fix 𝔍 𝔊 assume prems: 𝔊 : 𝔍 ↦↦_Cα 𝔄
  then interpret 𝔊: is-functor α 𝔍 𝔄 𝔊 .
  show cf-preserves-limits α 𝔉 𝔊
  proof(intro cf-preserves-limitsI)
    fix a σ assume σ : a <_CF.lim 𝔊 : 𝔍 ↦↦_Cα 𝔄
    then interpret σ: is-cat-limit α 𝔍 𝔄 𝔊 a σ .
```

show $\mathfrak{F} \circ_{CF-NTCF} \sigma : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) <_{CF.lim} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof(*intro is-cat-limitI*)
fix $r'\ \tau$ **assume** $\tau : r' <_{CF.cone} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
then interpret τ: *is-cat-cone* $\alpha\ r'\ \mathfrak{J}\ \mathfrak{B}\ \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle\ \tau$ **.**
note [*cat-cs-simps*] = *cf-comp-assoc-helper*[
where $\mathfrak{H}$=⟨*inv-cf* $\mathfrak{F}$⟩ **and** $\mathfrak{G}$=$\mathfrak{F}$ **and** $\mathfrak{F}$=$\mathfrak{G}$ **and** $\mathcal{Q}$=⟨*cf-id* $\mathfrak{A}$⟩
]
have *inv-τ*: *inv-cf* $\mathfrak{F} \circ_{CF-NTCF} \tau$:
inv-cf $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r' \rfloor) <_{CF.cone} \mathfrak{G} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by
(
cs-concl
cs-simp: *cat-cs-simps cf-cs-simps*
cs-intro: *cat-cs-intros cf-cs-intros*
)
from *is-cat-limit.cat-lim-unique-cone'*[*OF σ.is-cat-limit-axioms inv-τ*]
obtain f **where** *f*: f : *inv-cf* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r' \rfloor) \mapsto_{\mathfrak{A}} a$
and *f-up*: $\bigwedge j.\ j \in_\circ \mathfrak{J}(\lfloor Obj \rfloor) \Longrightarrow$
(*inv-cf* $\mathfrak{F} \circ_{CF-NTCF} \tau$)$(\lfloor NTMap \rfloor)(\lfloor j \rfloor) = \sigma(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} f$
and *f-unique*:
[[
f' : *inv-cf* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r' \rfloor) \mapsto_{\mathfrak{A}} a$;
$\bigwedge j.\ j \in_\circ \mathfrak{J}(\lfloor Obj \rfloor) \Longrightarrow$
(*inv-cf* $\mathfrak{F} \circ_{CF-NTCF} \tau$)$(\lfloor NTMap \rfloor)(\lfloor j \rfloor) = \sigma(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} f'$
]] $\Longrightarrow f' = f$
for f'
by *metis*
have [*cat-cs-simps*]: $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor \sigma(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \tau(\lfloor NTMap \rfloor)(\lfloor j \rfloor)$
if $j \in_\circ \mathfrak{J}(\lfloor Obj \rfloor)$ **for** j
proof–
from *f-up*[*OF that*] *that* **have**
inv-cf $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor \tau(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \rfloor) = \sigma(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} f$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor$*inv-cf* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor \tau(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \rfloor) \rfloor) =$
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor \sigma(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} f \rfloor)$
by *simp*
from *this that f* **show** *?thesis*
by
(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps cf-cs-simps* **cs-intro**: *cat-cs-intros*
)
simp
qed
show $\exists!f'.$
f' : $r' \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \wedge \tau = \mathfrak{F} \circ_{CF-NTCF} \sigma \bullet_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{B}\ f'$
proof(*intro ex1I conjI*; (*elim conjE*)*?*)
from f **have**
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor$*inv-cf* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r' \rfloor) \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
then show $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : r' \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
by (*cs-prems* **cs-shallow cs-simp**: *cf-cs-simps* **cs-intro**: *cat-cs-intros*)
show $\tau = \mathfrak{F} \circ_{CF-NTCF} \sigma \bullet_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{B}\ (\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor))$
proof(*rule ntcf-eqI*, *rule τ.is-ntcf-axioms*)
from f **show** $\mathfrak{F} \circ_{CF-NTCF} \sigma \bullet_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{B}\ (\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor))$:
cf-const $\mathfrak{J}\ \mathfrak{B}\ r' \mapsto_{CF} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by

```
        (
          cs-concl
            cs-simp: cat-cs-simps cf-cs-simps cs-intro: cat-cs-intros
        )
      then have dom-rhs:
        𝒟∘ ((𝔉 ∘CF−NTCF σ ⋆NTCF ntcf-const 𝔍 𝔅 (𝔉⦇ArrMap⦈⦇f⦈))⦇NTMap⦈) =
        𝔍⦇Obj⦈
        by (cs-concl cs-simp: cat-cs-simps)
      show
        τ⦇NTMap⦈ = (𝔉 ∘CF−NTCF σ ⋆NTCF ntcf-const 𝔍 𝔅 (𝔉⦇ArrMap⦈⦇f⦈))⦇NTMap⦈
      proof(rule vsv-eqI, unfold τ.ntcf-NTMap-vdomain dom-rhs)
        fix j assume j ∈∘ 𝔍⦇Obj⦈
        with f show τ⦇NTMap⦈⦇j⦈ =
          (𝔉 ∘CF−NTCF σ ⋆NTCF ntcf-const 𝔍 𝔅 (𝔉⦇ArrMap⦈⦇f⦈))⦇NTMap⦈⦇j⦈
          by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      qed (cs-concl cs-intro: V-cs-intros cat-cs-intros)+
    qed simp-all
    fix f′ assume prems′:
      f′ : r′ ↦𝔅 𝔉⦇ObjMap⦈⦇a⦈
      τ = 𝔉 ∘CF−NTCF σ ⋆NTCF ntcf-const 𝔍 𝔅 f′
    have τj-def: τ⦇NTMap⦈⦇j⦈ = 𝔉⦇ArrMap⦈⦇σ⦇NTMap⦈⦇j⦈⦈ ∘A𝔅 f′
      if j ∈∘ 𝔍⦇Obj⦈ for j
    proof–
      from prems′(2) have
        τ⦇NTMap⦈⦇j⦈ = (𝔉 ∘CF−NTCF σ ⋆NTCF ntcf-const 𝔍 𝔅 f′)⦇NTMap⦈⦇j⦈
        by simp
      from this prems′(1) that show ?thesis
        by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed
    have inv-cf 𝔉⦇ArrMap⦈⦇f′⦈ = f
    proof(rule f-unique)
      from prems′(1) show
        inv-cf 𝔉⦇ArrMap⦈⦇f′⦈ : inv-cf 𝔉⦇ObjMap⦈⦇r′⦈ ↦𝔄 a
        by
          (
            cs-concl
              cs-simp: cf-cs-simps cs-intro: cat-cs-intros cf-cs-intros
          )
      fix j assume j ∈∘ 𝔍⦇Obj⦈
      from this prems′(1) show
        (inv-cf 𝔉 ∘CF−NTCF τ)⦇NTMap⦈⦇j⦈ =
        σ⦇NTMap⦈⦇j⦈ ∘A𝔄 inv-cf 𝔉⦇ArrMap⦈⦇f′⦈
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cf-cs-simps τj-def
              cs-intro: cat-cs-intros cf-cs-intros
          )
    qed
    then have 𝔉⦇ArrMap⦈⦇inv-cf 𝔉⦇ArrMap⦈⦇f′⦈⦈ = 𝔉⦇ArrMap⦈⦇f⦈ by simp
    from this prems′(1) show f′ = 𝔉⦇ArrMap⦈⦇f⦈
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cf-cs-simps cs-intro: cat-cs-intros
        )
  qed
qed (cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros)
```

qed (*intro prems is-functor-axioms*)+
qed (*rule is-functor-axioms*)

lemma (**in** *is-iso-functor*) *iso-cf-is-cf-cocontinuous*: *is-cf-cocontinuous* α $\mathfrak{F}$
using *is-iso-functor.iso-cf-is-cf-continuous*[*OF is-iso-functor-op*]
by (*cs-prems* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*)

3.8 Tiny-continuous and tiny-cocontinuous functor

3.8.1 Definition and elementary properties

definition *is-tm-cf-continuous* :: $V \Rightarrow V \Rightarrow$ *bool*
where *is-tm-cf-continuous* α $\mathfrak{G}$ =
($\forall \mathfrak{F}$ $\mathfrak{J}$. $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{G}(|HomDom|) \longrightarrow$ *cf-preserves-limits* α $\mathfrak{G}$ $\mathfrak{F}$)

Rules.

context
fixes α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{F}$
assumes $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
begin

interpretation $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)

mk-ide rf *is-tm-cf-continuous-def*[**where** α=α **and** $\mathfrak{G}$=$\mathfrak{G}$, *unfolded cat-cs-simps*]
|*intro is-tm-cf-continuousI*|
|*dest is-tm-cf-continuousD'*|
|*elim is-tm-cf-continuousE'*|

end

lemmas *is-tm-cf-continuousD*[*dest*!] = *is-tm-cf-continuousD'*[*rotated*]
and *is-tm-cf-continuousE*[*elim*!] = *is-tm-cf-continuousE'*[*rotated*]

Elementary properties.

lemma (**in** *is-functor*) *cf-continuous-is-tm-cf-continuous*:
assumes *is-cf-continuous* α $\mathfrak{F}$
shows *is-tm-cf-continuous* α $\mathfrak{F}$
proof(*intro is-tm-cf-continuousI*, *rule is-functor-axioms*)
fix $\mathfrak{F}'$ $\mathfrak{J}$ **assume** $\mathfrak{F}' : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{A}$
then interpret $\mathfrak{F}'$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{A}$ $\mathfrak{F}'$**.**
show *cf-preserves-limits* α $\mathfrak{F}$ $\mathfrak{F}'$
by
(
intro is-cf-continuousD[*OF assms*(*1*) - *is-functor-axioms*],
rule $\mathfrak{F}'$*.is-functor-axioms*
)
qed

4 Initial and terminal objects as limits and colimits

4.1 Initial and terminal objects as limits/colimits of an empty diagram

4.1.1 Definition and elementary properties

See [1][2], [1][3] and Chapter X-1 in [9].

locale *is-cat-obj-empty-terminal* = *is-cat-limit* α *cat-0* $\mathfrak{C}$ ‹*cf-0* $\mathfrak{C}$› z $\mathfrak{Z}$
  for α $\mathfrak{C}$ z $\mathfrak{Z}$

syntax *-is-cat-obj-empty-terminal* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (‹(- :/ - <$_{CF.1}$ 0_{CF} :/ 0_C $\mapsto\mapsto_{C^1}$ -)› [51, 51] 51)
syntax-consts *-is-cat-obj-empty-terminal* $\rightleftharpoons$ *is-cat-obj-empty-terminal*
translations $\mathfrak{Z}$: z <$_{CF.1}$ 0_{CF} : 0_C $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$ $\rightleftharpoons$
  CONST is-cat-obj-empty-terminal α $\mathfrak{C}$ z $\mathfrak{Z}$

locale *is-cat-obj-empty-initial* = *is-cat-colimit* α *cat-0* $\mathfrak{C}$ ‹*cf-0* $\mathfrak{C}$› z $\mathfrak{Z}$
  for α $\mathfrak{C}$ z $\mathfrak{Z}$

syntax *-is-cat-obj-empty-initial* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (‹(- :/ 0_{CF} >$_{CF.0}$ - :/ 0_C $\mapsto\mapsto_{C^1}$ -)› [51, 51] 51)
syntax-consts *-is-cat-obj-empty-initial* $\rightleftharpoons$ *is-cat-obj-empty-initial*
translations $\mathfrak{Z}$: 0_{CF} >$_{CF.0}$ z : 0_C $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$ $\rightleftharpoons$
  CONST is-cat-obj-empty-initial α $\mathfrak{C}$ z $\mathfrak{Z}$

Rules.

lemma (**in** *is-cat-obj-empty-terminal*)
  is-cat-obj-empty-terminal-axioms'[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows $\mathfrak{Z}$: z' <$_{CF.1}$ 0_{CF} : 0_C $\mapsto\mapsto_{C_{\alpha'}}$ $\mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-obj-empty-terminal-axioms*)

mk-ide rf *is-cat-obj-empty-terminal-def*
  |*intro is-cat-obj-empty-terminalI*|
  |*dest is-cat-obj-empty-terminalD*[*dest*]|
  |*elim is-cat-obj-empty-terminalE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-empty-terminalD*

lemma (**in** *is-cat-obj-empty-initial*)
  is-cat-obj-empty-initial-axioms'[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows $\mathfrak{Z}$: 0_{CF} >$_{CF.0}$ z' : 0_C $\mapsto\mapsto_{C_{\alpha'}}$ $\mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-obj-empty-initial-axioms*)

mk-ide rf *is-cat-obj-empty-initial-def*
  |*intro is-cat-obj-empty-initialI*|
  |*dest is-cat-obj-empty-initialD*[*dest*]|
  |*elim is-cat-obj-empty-initialE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-empty-initialD*

Duality.

lemma (**in** *is-cat-obj-empty-terminal*) *is-cat-obj-empty-initial-op*:
  op-ntcf $\mathfrak{Z}$: 0_{CF} >$_{CF.0}$ z : 0_C $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$

[2] https://ncatlab.org/nlab/show/initial+object
[3] https://ncatlab.org/nlab/show/terminal+object

by (*intro is-cat-obj-empty-initialI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps op-cf-cf-0* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-obj-empty-terminal*) *is-cat-obj-empty-initial-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows *op-ntcf* $\mathfrak{Z} : 0_{CF} >_{CF.0} z : 0_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-empty-initial-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-empty-terminal.is-cat-obj-empty-initial-op'*

lemma (**in** *is-cat-obj-empty-initial*) *is-cat-obj-empty-terminal-op*:
op-ntcf $\mathfrak{Z} : z <_{CF.1} 0_{CF} : 0_C \mapsto\mapsto_{C\alpha} op\text{-}cat\ \mathfrak{C}$
by (*intro is-cat-obj-empty-terminalI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps op-cf-cf-0* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-obj-empty-initial*) *is-cat-obj-empty-terminal-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows *op-ntcf* $\mathfrak{Z} : z <_{CF.1} 0_{CF} : 0_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-empty-terminal-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-empty-initial.is-cat-obj-empty-terminal-op'*

Elementary properties.

lemma (**in** *is-cat-obj-empty-terminal*) *cat-oet-ntcf-0*: $\mathfrak{Z} = ntcf\text{-}0\ \mathfrak{C}$
by (*rule is-ntcf-is-ntcf-0-if-cat-0*)
(*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma (**in** *is-cat-obj-empty-initial*) *cat-oei-ntcf-0*: $\mathfrak{Z} = ntcf\text{-}0\ \mathfrak{C}$
by (*rule is-ntcf-is-ntcf-0-if-cat-0*)
(*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

4.1.2 Initial and terminal objects as limits/colimits of an empty diagram are initial and terminal objects

lemma (**in** *category*) *cat-obj-terminal-is-cat-obj-empty-terminal*:
assumes *obj-terminal* $\mathfrak{C}\ z$
shows *ntcf-0* $\mathfrak{C} : z <_{CF.1} 0_{CF} : 0_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

from *assms* **have** *z*: $z \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
from *z* **have** [*cat-cs-simps*]: *cf-const cat-0* $\mathfrak{C}\ z = cf\text{-}0\ \mathfrak{C}$
by (*intro is-functor-is-cf-0-if-cat-0*) (*cs-concl* **cs-intro**: *cat-cs-intros*)
note *obj-terminalD* = *obj-terminalD*[*OF assms*]

show *?thesis*
proof
(
intro is-cat-obj-empty-terminalI is-cat-limitI is-cat-coneI,
unfold cat-cs-simps
)
show $\exists!f'.\ f' : r' \mapsto_{\mathfrak{C}} z \wedge u' = ntcf\text{-}0\ \mathfrak{C} \bullet_{NTCF} ntcf\text{-}const\ cat\text{-}0\ \mathfrak{C}\ f'$
if $u' : r' <_{CF.cone} cf\text{-}0\ \mathfrak{C} : cat\text{-}0 \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **for** $u'\ r'$

```
    proof–
      interpret u′: is-cat-cone α r′ cat-0 𝔈 ‹cf-0 𝔈› u′ by (rule that)
      from z have [cat-cs-simps]: cf-const cat-0 𝔈 r′ = cf-0 𝔈
        by (intro is-functor-is-cf-0-if-cat-0)
          (cs-concl cs-shallow cs-intro: cat-cs-intros)
      have u′-def: u′ = ntcf-0 𝔈
        by
          (
            rule is-ntcf-is-ntcf-0-if-cat-0[
              OF u′.is-ntcf-axioms, unfolded cat-cs-simps
              ]
          )
      from obj-terminalD(2)[OF u′.cat-cone-obj] obtain f′
        where f′: f′ : r′ ↦_𝔈 z
          and f′-unique: f′′ : r′ ↦_𝔈 z ⟹ f′′ = f′
        for f′′
        by auto
      from f′ have [cat-cs-simps]: ntcf-const cat-0 𝔈 f′ = ntcf-0 𝔈
        by (intro is-ntcf-is-ntcf-0-if-cat-0(1))
          (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      show ?thesis
      proof(intro ex1I conjI; (elim conjE)?)
        show u′ = ntcf-0 𝔈 ∘_NTCF ntcf-const cat-0 𝔈 f′
          by
            (
              cs-concl cs-shallow
                cs-simp: u′-def cat-cs-simps cs-intro: cat-cs-intros
            )
        fix f′′ assume prems:
          f′′ : r′ ↦_𝔈 z u′ = ntcf-0 𝔈 ∘_NTCF ntcf-const cat-0 𝔈 f′′
        show f′′ = f′ by (rule f′-unique[OF prems(1)])
      qed (rule f′)
    qed
  qed (cs-concl cs-simp: cat-cs-simps cs-intro: z cat-cs-intros)

qed

lemma (in category) cat-obj-initial-is-cat-obj-empty-initial:
  assumes obj-initial 𝔈 z
  shows ntcf-0 𝔈 : 0_CF >_CF.0 z : 0_C ↦↦_Cα 𝔈
proof–
  have z: obj-terminal (op-cat 𝔈) z unfolding cat-op-simps by (rule assms)
  show ?thesis
    by
      (
        rule is-cat-obj-empty-terminal.is-cat-obj-empty-initial-op
          [
            OF category.cat-obj-terminal-is-cat-obj-empty-terminal[
              OF category-op z, folded op-ntcf-ntcf-0
              ],
            unfolded cat-op-simps op-ntcf-ntcf-0
          ]
      )
qed

lemma (in is-cat-obj-empty-terminal) cat-oet-obj-terminal: obj-terminal 𝔈 z
proof–
  show obj-terminal 𝔈 z
```

```
  proof(rule obj-terminalI)
    fix a assume prems: a ∈∘ ℭ⦇Obj⦈
    have [cat-cs-simps]: cf-const cat-0 ℭ a = cf-0 ℭ
      by (rule is-functor-is-cf-0-if-cat-0)
        (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros prems)
    from prems have ntcf-0 ℭ : a <CF.cone cf-0 ℭ : cat-0 ↦↦Cα ℭ
      by (intro is-cat-coneI)
        (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from cat-lim-ua-fo[OF this] obtain f'
      where f': f' : a ↦ℭ z
        and ntcf-0 ℭ = ℨ ∘NTCF ntcf-const cat-0 ℭ f'
        and f'-unique:
          ⟦ f'' : a ↦ℭ z; ntcf-0 ℭ = ℨ ∘NTCF ntcf-const cat-0 ℭ f'' ⟧ ⟹
            f'' = f'
      for f''
      by metis
    show ∃!f'. f' : a ↦ℭ z
    proof(intro ex1I)
      fix f'' assume prems': f'' : a ↦ℭ z
      from prems' have ntcf-0 ℭ = ntcf-0 ℭ ∘NTCF ntcf-const cat-0 ℭ f''
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      from f'-unique[OF prems', unfolded cat-oet-ntcf-0, OF this]
      show f'' = f'.
    qed (rule f')
  qed (rule cat-cone-obj)
qed

lemma (in is-cat-obj-empty-initial) cat-oei-obj-initial: obj-initial ℭ z
  by
    (
      rule is-cat-obj-empty-terminal.cat-oet-obj-terminal[
        OF is-cat-obj-empty-initial.is-cat-obj-empty-terminal-op[
          OF is-cat-obj-empty-initial-axioms
          ],
        unfolded cat-op-simps
        ]
    )

lemma (in category) cat-is-cat-obj-empty-terminal-obj-terminal-iff:
  (ntcf-0 ℭ : z <CF.1 0CF : 0C ↦↦Cα ℭ) ⟷ obj-terminal ℭ z
  using
    cat-obj-terminal-is-cat-obj-empty-terminal
    is-cat-obj-empty-terminal.cat-oet-obj-terminal
  by auto

lemma (in category) cat-is-cat-obj-empty-initial-obj-initial-iff:
  (ntcf-0 ℭ : 0CF >CF.0 z : 0C ↦↦Cα ℭ) ⟷ obj-initial ℭ z
  using
    cat-obj-initial-is-cat-obj-empty-initial
    is-cat-obj-empty-initial.cat-oei-obj-initial
  by auto
```

4.2 Initial cone and terminal cocone

4.2.1 Definitions and elementary properties

```
definition ntcf-initial :: V ⇒ V ⇒ V
  where ntcf-initial ℭ z =
```

```
    [
      (λb∈₀𝔈(|Obj|). THE f. f : z ↦_𝔈 b),
      cf-const 𝔈 𝔈 z,
      cf-id 𝔈,
      𝔈,
      𝔈
    ]₀

definition ntcf-terminal :: V ⇒ V ⇒ V
  where ntcf-terminal 𝔈 z =
    [
      (λb∈₀𝔈(|Obj|). THE f. f : b ↦_𝔈 z),
      cf-id 𝔈,
      cf-const 𝔈 𝔈 z,
      𝔈,
      𝔈
    ]₀
```

Components.

```
lemma ntcf-initial-components:
  shows ntcf-initial 𝔈 z(|NTMap|) = (λc∈₀𝔈(|Obj|). THE f. f : z ↦_𝔈 c)
    and ntcf-initial 𝔈 z(|NTDom|) = cf-const 𝔈 𝔈 z
    and ntcf-initial 𝔈 z(|NTCod|) = cf-id 𝔈
    and ntcf-initial 𝔈 z(|NTDGDom|) = 𝔈
    and ntcf-initial 𝔈 z(|NTDGCod|) = 𝔈
  unfolding ntcf-initial-def nt-field-simps
  by (simp-all add: nat-omega-simps)

lemmas [cat-lim-cs-simps] = ntcf-initial-components(2–5)

lemma ntcf-terminal-components:
  shows ntcf-terminal 𝔈 z(|NTMap|) = (λc∈₀𝔈(|Obj|). THE f. f : c ↦_𝔈 z)
    and ntcf-terminal 𝔈 z(|NTDom|) = cf-id 𝔈
    and ntcf-terminal 𝔈 z(|NTCod|) = cf-const 𝔈 𝔈 z
    and ntcf-terminal 𝔈 z(|NTDGDom|) = 𝔈
    and ntcf-terminal 𝔈 z(|NTDGCod|) = 𝔈
  unfolding ntcf-terminal-def nt-field-simps
  by (simp-all add: nat-omega-simps)

lemmas [cat-lim-cs-simps] = ntcf-terminal-components(2–5)
```

Duality.

```
lemma ntcf-initial-op[cat-op-simps]:
  op-ntcf (ntcf-initial 𝔈 z) = ntcf-terminal (op-cat 𝔈) z
  unfolding
    ntcf-initial-def ntcf-terminal-def op-ntcf-def
    nt-field-simps cat-op-simps
  by (auto simp: nat-omega-simps cat-op-simps)

lemma ntcf-cone-terminal-op[cat-op-simps]:
  op-ntcf (ntcf-terminal 𝔈 z) = ntcf-initial (op-cat 𝔈) z
  unfolding
    ntcf-initial-def ntcf-terminal-def op-ntcf-def
    nt-field-simps cat-op-simps
  by (auto simp: nat-omega-simps cat-op-simps)
```

4.2.2 Natural transformation map

mk-VLambda *ntcf-initial-components*(*1*)
|*vsv ntcf-initial-vsv*[*cat-lim-cs-intros*]|
|*vdomain ntcf-initial-vdomain*[*cat-lim-cs-simps*]|
|*app ntcf-initial-app*|

mk-VLambda *ntcf-terminal-components*(*1*)
|*vsv ntcf-terminal-vsv*[*cat-lim-cs-intros*]|
|*vdomain ntcf-terminal-vdomain*[*cat-lim-cs-simps*]|
|*app ntcf-terminal-app*|

lemma (**in** *category*)
assumes *obj-initial* $\mathfrak{C}$ *z* **and** $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-initial-NTMap-app-is-arr*:
ntcf-initial $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!) : z \mapsto_{\mathfrak{C}} c$
and *ntcf-initial-NTMap-app-unique*:
$\bigwedge f'.\ f' : z \mapsto_{\mathfrak{C}} c \Longrightarrow f' =$ *ntcf-initial* $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!)$
proof–
from *obj-initialD*(*2*)[*OF assms*(*1,2*)] **obtain** *f*
where *f*: $f : z \mapsto_{\mathfrak{C}} c$
and *f-unique*: $f' : z \mapsto_{\mathfrak{C}} c \Longrightarrow f' = f$
for f'
by *auto*
show *is-arr*: *ntcf-initial* $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!) : z \mapsto_{\mathfrak{C}} c$
proof(*cs-concl-step ntcf-initial-app*, *rule assms*(*2*), *rule theI*)
fix f' **assume** $f' : z \mapsto_{\mathfrak{C}} c$
from *f-unique*[*OF this*] **show** $f' = f$**.**
qed (*rule f*)
fix f' **assume** $f' : z \mapsto_{\mathfrak{C}} c$
from *f-unique*[*OF this*, *folded f-unique*[*OF is-arr*]]
show $f' =$ *ntcf-initial* $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!)$**.**
qed

lemma (**in** *category*) *ntcf-initial-NTMap-app-is-arr'*[*cat-lim-cs-intros*]:
assumes *obj-initial* $\mathfrak{C}$ *z*
and $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $\mathfrak{C}' = \mathfrak{C}$
and $z' = z$
and $c' = c$
shows *ntcf-initial* $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!) : z' \mapsto_{\mathfrak{C}'} c'$
using *assms*(*1,2*)
unfolding *assms*(*3–5*)
by (*rule ntcf-initial-NTMap-app-is-arr*)

lemma (**in** *category*)
assumes *obj-terminal* $\mathfrak{C}$ *z* **and** $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-terminal-NTMap-app-is-arr*:
ntcf-terminal $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!) : c \mapsto_{\mathfrak{C}} z$
and *ntcf-terminal-NTMap-app-unique*:
$\bigwedge f'.\ f' : c \mapsto_{\mathfrak{C}} z \Longrightarrow f' =$ *ntcf-terminal* $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!)$
proof–
from *obj-terminalD*(*2*)[*OF assms*(*1,2*)] **obtain** *f*
where *f*: $f : c \mapsto_{\mathfrak{C}} z$
and *f-unique*: $f' : c \mapsto_{\mathfrak{C}} z \Longrightarrow f' = f$
for f'
by *auto*
show *is-arr*: *ntcf-terminal* $\mathfrak{C}$ *z*$(\!|NTMap|\!)(\!|c|\!) : c \mapsto_{\mathfrak{C}} z$

proof(*cs-concl-step ntcf-terminal-app*, *rule assms*(*2*), *rule theI*)
fix f' **assume** $f' : c \mapsto_{\mathfrak{C}} z$
from *f-unique*[*OF this*] **show** $f' = f$.
qed (*rule f*)
fix f' **assume** $f' : c \mapsto_{\mathfrak{C}} z$
from *f-unique*[*OF this*, *folded f-unique*[*OF is-arr*]]
show $f' =$ *ntcf-terminal* $\mathfrak{C}$ z(|*NTMap*|)(|c|).
qed

lemma (**in** *category*) *ntcf-terminal-NTMap-app-is-arr'*[*cat-lim-cs-intros*]:
assumes *obj-terminal* $\mathfrak{C}$ z
and $c \in_{\circ} \mathfrak{C}$(|*Obj*|)
and $\mathfrak{C}' = \mathfrak{C}$
and $z' = z$
and $c' = c$
shows *ntcf-terminal* $\mathfrak{C}$ z(|*NTMap*|)(|c|) : $c' \mapsto_{\mathfrak{C}'} z'$
using *assms*(*1*,*2*)
unfolding *assms*(*3–5*)
by (*rule ntcf-terminal-NTMap-app-is-arr*)

4.3 Initial and terminal objects as limits/colimits of the identity functor

4.3.1 Definition and elementary properties

See [1][4], [1][5] and Chapter X-1 in [9].

locale *is-cat-obj-id-initial* = *is-cat-limit* α $\mathfrak{C}$ $\mathfrak{C}$ ‹*cf-id* $\mathfrak{C}$› z $\mathfrak{Z}$ **for** α $\mathfrak{C}$ z $\mathfrak{Z}$

syntax *-is-cat-obj-id-initial* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - <$_{CF.0}$ id_C :/ ↦↦$_{C1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-cat-obj-id-initial* ⇌ *is-cat-obj-id-initial*
translations $\mathfrak{Z}$: z <$_{CF.0}$ id_C : ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌
CONST is-cat-obj-id-initial α $\mathfrak{C}$ z $\mathfrak{Z}$

locale *is-cat-obj-id-terminal* = *is-cat-colimit* α $\mathfrak{C}$ $\mathfrak{C}$ ‹*cf-id* $\mathfrak{C}$› z $\mathfrak{Z}$ **for** α $\mathfrak{C}$ z $\mathfrak{Z}$

syntax *-is-cat-obj-id-terminal* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ id_C >$_{CF.1}$ - :/ ↦↦$_{C1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-cat-obj-id-terminal* ⇌ *is-cat-obj-id-terminal*
translations $\mathfrak{Z}$: id_C >$_{CF.1}$ z : ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌
CONST is-cat-obj-id-terminal α $\mathfrak{C}$ z $\mathfrak{Z}$

Rules.

lemma (**in** *is-cat-obj-id-initial*)
is-cat-obj-id-initial-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{Z}$: z' <$_{CF.0}$ id_C : ↦↦$_{C\alpha}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-id-initial-axioms*)

mk-ide rf *is-cat-obj-id-initial-def*
|*intro is-cat-obj-id-initialI*|
|*dest is-cat-obj-id-initialD*[*dest*]|
|*elim is-cat-obj-id-initialE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-id-initialD*

[4] https://ncatlab.org/nlab/show/initial+object
[5] https://ncatlab.org/nlab/show/terminal+object

lemma (**in** *is-cat-obj-id-terminal*)
is-cat-obj-id-terminal-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{Z} : id_C >_{CF.1} z' : \mapsto\!\!\mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-id-terminal-axioms*)

mk-ide rf *is-cat-obj-id-terminal-def*
|*intro is-cat-obj-id-terminalI*|
|*dest is-cat-obj-id-terminalD*[*dest*]|
|*elim is-cat-obj-id-terminalE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-id-terminalD*

Duality.

lemma (**in** *is-cat-obj-id-initial*) *is-cat-obj-id-terminal-op*:
op-ntcf $\mathfrak{Z} : id_C >_{CF.1} z : \mapsto\!\!\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-obj-id-terminalI*)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)

lemma (**in** *is-cat-obj-id-initial*) *is-cat-obj-id-terminal-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\mathfrak{Z} : id_C >_{CF.1} z : \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-id-terminal-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-id-initial.is-cat-obj-id-terminal-op'*

lemma (**in** *is-cat-obj-id-terminal*) *is-cat-obj-id-initial-op*:
op-ntcf $\mathfrak{Z} : z <_{CF.0} id_C : \mapsto\!\!\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-obj-id-initialI*)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)

lemma (**in** *is-cat-obj-id-terminal*) *is-cat-obj-id-initial-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\mathfrak{Z} : z <_{CF.0} id_C : \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-id-initial-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-id-terminal.is-cat-obj-id-initial-op'*

4.3.2 Initial and terminal objects as limits/colimits are initial and terminal objects

lemma (**in** *category*) *cat-obj-initial-is-cat-obj-id-initial*:
assumes *obj-initial* $\mathfrak{C}$ z
shows *ntcf-initial* $\mathfrak{C}$ z : $z <_{CF.0} id_C : \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-cat-obj-id-initialI is-cat-limitI*)

from *assms* **have** *z*: $z \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
note *obj-initialD* = *obj-initialD*[*OF assms*]

show *ntcf-initial* $\mathfrak{C}$ z : $z <_{CF.cone}$ *cf-id* $\mathfrak{C}$: $\mathfrak{C} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-cat-coneI is-ntcfI'*, *unfold cat-lim-cs-simps*)
show *vfsequence* (*ntcf-initial* $\mathfrak{C}$ z)
unfolding *ntcf-initial-def* **by** *auto*
show *vcard* (*ntcf-initial* $\mathfrak{C}$ z) = $5_{\mathbb{N}}$
unfolding *ntcf-initial-def* **by** (*simp add*: *nat-omega-simps*)
show *ntcf-initial* $\mathfrak{C}$ z(|*NTMap*|)(|a|) :
cf-const $\mathfrak{C}$ $\mathfrak{C}$ z(|*ObjMap*|)(|a|) $\mapsto_{\mathfrak{C}}$ *cf-id* $\mathfrak{C}$(|*ObjMap*|)(|a|)
if $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **for** a
using *that assms*(*1*)

```
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-lim-cs-intros
       )
  show
    ntcf-initial 𝔈 z(⁅NTMap⁆)(⁅b⁆) ∘_{A𝔈} cf-const 𝔈 𝔈 z(⁅ArrMap⁆)(⁅f⁆) =
      cf-id 𝔈(⁅ArrMap⁆)(⁅f⁆) ∘_{A𝔈} ntcf-initial 𝔈 z(⁅NTMap⁆)(⁅a⁆)
    if f : a ↦_𝔈 b for a b f
  proof–
    from that assms(1) have
      f ∘_{A𝔈} ntcf-initial 𝔈 z(⁅NTMap⁆)(⁅a⁆) : z ↦_𝔈 b
      by (cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros)
    note [cat-cs-simps] = ntcf-initial-NTMap-app-unique[
        OF assms(1) cat-is-arrD(3)[OF that] this
        ]
    from that assms(1) show ?thesis
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-lim-cs-intros
        )
  qed
qed (use z in ‹cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros›)+
then interpret i: is-cat-cone α z 𝔈 𝔈 ‹cf-id 𝔈› ‹ntcf-initial 𝔈 z› .

fix u r assume u : r <_{CF.cone} cf-id 𝔈 : 𝔈 ↦↦_{Cα} 𝔈
then interpret u: is-cat-cone α r 𝔈 𝔈 ‹cf-id 𝔈› u .

from obj-initialD(2)[OF u.cat-cone-obj] obtain f
  where f: f : z ↦_𝔈 r and f-unique: f' : z ↦_𝔈 r ⟹ f' = f for f'
  by auto
note u.cat-cone-Comp-commute[cat-cs-simps del]
from u.ntcf-Comp-commute[OF f] f have u(⁅NTMap⁆)(⁅r⁆) = f ∘_{A𝔈} u(⁅NTMap⁆)(⁅z⁆)
  by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

show ∃!f'.
  f' : r ↦_𝔈 z ∧
  u = ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 f'
proof(intro ex1I conjI; (elim conjE)?)
  from f show u(⁅NTMap⁆)(⁅z⁆) : r ↦_𝔈 z
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show u = ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 (u(⁅NTMap⁆)(⁅z⁆))
  proof(rule ntcf-eqI, rule u.is-ntcf-axioms)
    show ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 (u(⁅NTMap⁆)(⁅z⁆)) :
      cf-const 𝔈 𝔈 r ↦_{CF} cf-id 𝔈 : 𝔈 ↦↦_{Cα} 𝔈
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from z have dom-rhs:
      𝒟∘ ((ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 (u(⁅NTMap⁆)(⁅z⁆)))(⁅NTMap⁆)) =
        𝔈(⁅Obj⁆)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show u(⁅NTMap⁆) =
      (ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 (u(⁅NTMap⁆)(⁅z⁆)))(⁅NTMap⁆)
    proof(rule vsv-eqI, unfold dom-rhs u.ntcf-NTMap-vdomain)
      fix c assume prems: c ∈∘ 𝔈(⁅Obj⁆)
      then have ic: ntcf-initial 𝔈 z(⁅NTMap⁆)(⁅c⁆) : z ↦_𝔈 c
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      from u.ntcf-Comp-commute[OF ic] ic have [cat-cs-simps]:
```

```
      ntcf-initial 𝔈 z(⁅NTMap⁆)(⁅c⁆) ∘_{A𝔈} u(⁅NTMap⁆)(⁅z⁆) = u(⁅NTMap⁆)(⁅c⁆)
      by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros) simp
    from prems z show u(⁅NTMap⁆)(⁅c⁆) =
      (ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 (u(⁅NTMap⁆)(⁅z⁆)))(⁅NTMap⁆)(⁅c⁆)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed (auto intro: cat-cs-intros)
 qed simp-all
 fix f' assume prems:
  f' : r ↦_𝔈 z
  u = ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 f'
 from z have ntcf-initial 𝔈 z(⁅NTMap⁆)(⁅z⁆) : z ↦_𝔈 z
  by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 note [cat-cs-simps] = cat-obj-initial-CId[OF assms this, symmetric]
 from prems(2) have
  u(⁅NTMap⁆)(⁅z⁆) = (ntcf-initial 𝔈 z ∙_{NTCF} ntcf-const 𝔈 𝔈 f')(⁅NTMap⁆)(⁅z⁆)
  by simp
 from this prems(1) show f' = u(⁅NTMap⁆)(⁅z⁆)
  by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros) simp
 qed
qed

lemma (in category) cat-obj-terminal-is-cat-obj-id-terminal:
 assumes obj-terminal 𝔈 z
 shows ntcf-terminal 𝔈 z : id_C >_{CF.1} z : ↦↦_{Cα} 𝔈
 by
  (
   rule is-cat-obj-id-initial.is-cat-obj-id-terminal-op
    [
     OF category.cat-obj-initial-is-cat-obj-id-initial[
      OF category-op op-cat-obj-initial[THEN iffD2, OF assms(1)]
      ],
     unfolded cat-op-simps
    ]
  )

lemma cat-cone-CId-obj-initial:
 assumes 𝔍 : z <_{CF.cone} cf-id 𝔈 : 𝔈 ↦↦_{Cα} 𝔈 and 𝔍(⁅NTMap⁆)(⁅z⁆) = 𝔈(⁅CId⁆)(⁅z⁆)
 shows obj-initial 𝔈 z
proof(intro obj-initialI)
 interpret 𝔍: is-cat-cone α z 𝔈 𝔈 ‹cf-id 𝔈› 𝔍 by (rule assms(1))
 show z ∈_∘ 𝔈(⁅Obj⁆) by (cs-concl cs-intro: cat-cs-intros)
 fix c assume prems: c ∈_∘ 𝔈(⁅Obj⁆)
 show ∃!f. f : z ↦_𝔈 c
 proof(intro ex1I)
  from prems show 𝔍c: 𝔍(⁅NTMap⁆)(⁅c⁆) : z ↦_𝔈 c
   by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  fix f assume prems': f : z ↦_𝔈 c
  from 𝔍.ntcf-Comp-commute[OF prems'] prems' 𝔍c show f = 𝔍(⁅NTMap⁆)(⁅c⁆)
   by (cs-prems cs-simp: cat-cs-simps assms(2) cs-intro: cat-cs-intros) simp
 qed
qed

lemma cat-cocone-CId-obj-terminal:
 assumes 𝔍 : cf-id 𝔈 >_{CF.cocone} z : 𝔈 ↦↦_{Cα} 𝔈 and 𝔍(⁅NTMap⁆)(⁅z⁆) = 𝔈(⁅CId⁆)(⁅z⁆)
 shows obj-terminal 𝔈 z
proof–
 interpret 𝔍: is-cat-cocone α z 𝔈 𝔈 ‹cf-id 𝔈› 𝔍 by (rule assms(1))
 show ?thesis
```

```
    by
      (
        rule cat-cone-CId-obj-initial
          [
            OF 𝔍.is-cat-cone-op[unfolded cat-op-simps],
            unfolded cat-op-simps,
            OF assms(2)
          ]
      )
qed

lemma (in is-cat-obj-id-initial) cat-oii-obj-initial: obj-initial 𝔈 z
proof(rule cat-cone-CId-obj-initial, rule is-cat-cone-axioms)
  from cat-lim-unique-cone'[OF is-cat-cone-axioms] obtain f
    where f: f : z ↦_𝔈 z
      and 𝔍'j: ⋀j. j ∈₀ 𝔈(|Obj|) ⟹ 𝔍(|NTMap|)(|j|) = 𝔍(|NTMap|)(|j|) ∘_A𝔈 f
      and f-unique:
        ⟦
          f' : z ↦_𝔈 z;
          ⋀j. j ∈₀ 𝔈(|Obj|) ⟹ 𝔍(|NTMap|)(|j|) = 𝔍(|NTMap|)(|j|) ∘_A𝔈 f'
        ⟧ ⟹ f' = f
    for f'
    by metis
  have CId-z: 𝔈(|CId|)(|z|) : z ↦_𝔈 z
    by (cs-concl cs-intro: cat-cs-intros)
  have 𝔍(|NTMap|)(|j|) = 𝔍(|NTMap|)(|j|) ∘_A𝔈 𝔈(|CId|)(|z|) if j ∈₀ 𝔈(|Obj|) for j
    using that by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from f-unique[OF CId-z this] have CId-f: 𝔈(|CId|)(|z|) = f .
  have 𝔍z: 𝔍(|NTMap|)(|z|) : z ↦_𝔈 z
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  have 𝔍(|NTMap|)(|c|) = 𝔍(|NTMap|)(|c|) ∘_A𝔈 𝔍(|NTMap|)(|z|) if c ∈₀ 𝔈(|Obj|) for c
  proof–
    from that have 𝔍c: 𝔍(|NTMap|)(|c|) : z ↦_𝔈 c
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    note cat-cone-Comp-commute[cat-cs-simps del]
    from ntcf-Comp-commute[OF 𝔍c] 𝔍c show
      𝔍(|NTMap|)(|c|) = 𝔍(|NTMap|)(|c|) ∘_A𝔈 𝔍(|NTMap|)(|z|)
      by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed
  from f-unique[OF 𝔍z this] have 𝔍(|NTMap|)(|z|) = f .
  with CId-f show 𝔍(|NTMap|)(|z|) = 𝔈(|CId|)(|z|) by simp
qed

lemma (in is-cat-obj-id-terminal) cat-oit-obj-terminal: obj-terminal 𝔈 z
  by
    (
      rule is-cat-obj-id-initial.cat-oii-obj-initial[
        OF is-cat-obj-id-initial-op, unfolded cat-op-simps
        ]
    )

lemma (in category) cat-is-cat-obj-id-initial-obj-initial-iff:
  (ntcf-initial 𝔈 z : z <_CF.0 id_C : ↦↦_Cα 𝔈) ⟷ obj-initial 𝔈 z
  using
    cat-obj-initial-is-cat-obj-id-initial
    is-cat-obj-id-initial.cat-oii-obj-initial
  by auto
```

lemma (**in** *category*) *cat-is-cat-obj-id-terminal-obj-terminal-iff*:
(*ntcf-terminal* $\mathfrak{C}$ z : id_C $>_{CF.1}$ z : $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$) $\longleftrightarrow$ *obj-terminal* $\mathfrak{C}$ z
using
cat-obj-terminal-is-cat-obj-id-terminal
is-cat-obj-id-terminal.cat-oit-obj-terminal
by *auto*

5 Products and coproducts as limits and colimits

5.1 Product and coproduct

5.1.1 Definition and elementary properties

The definition of the product object is a specialization of the definition presented in Chapter III-4 in [9]. In the definition presented below, the discrete category that is used in the definition presented in [9] is parameterized by an index set and the functor from the discrete category is parameterized by a function from the index set to the set of the objects of the category.

locale *is-cat-obj-prod* =
  is-cat-limit α ‹:$_C$ I› $\mathfrak{C}$ ‹:→: I A $\mathfrak{C}$› P π + *cf-discrete* α I A $\mathfrak{C}$
  for α I A $\mathfrak{C}$ P π

syntax *-is-cat-obj-prod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (‹(- :/ - <$_{CF.\prod}$ - :/ - ↦↦$_{C^1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-obj-prod* ⇌ *is-cat-obj-prod*
translations π : P <$_{CF.\prod}$ A : I ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌
  CONST is-cat-obj-prod α I A $\mathfrak{C}$ P π

locale *is-cat-obj-coprod* =
  is-cat-colimit α ‹:$_C$ I› $\mathfrak{C}$ ‹:→: I A $\mathfrak{C}$› U π + *cf-discrete* α I A $\mathfrak{C}$
  for α I A $\mathfrak{C}$ U π

syntax *-is-cat-obj-coprod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (‹(- :/ - >$_{CF.\coprod}$ - :/ - ↦↦$_{C^1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-obj-coprod* ⇌ *is-cat-obj-coprod*
translations π : A >$_{CF.\coprod}$ U : I ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌
  CONST is-cat-obj-coprod α I A $\mathfrak{C}$ U π

Rules.

lemma (**in** *is-cat-obj-prod*) *is-cat-obj-prod-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows π : P' <$_{CF.\prod}$ A' : I' ↦↦$_{C_{\alpha'}}$ $\mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-obj-prod-axioms*)

mk-ide rf *is-cat-obj-prod-def*
  |*intro is-cat-obj-prodI*|
  |*dest is-cat-obj-prodD*[*dest*]|
  |*elim is-cat-obj-prodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-prodD*

lemma (**in** *is-cat-obj-coprod*) *is-cat-obj-coprod-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $U' = U$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows π : A' >$_{CF.\coprod}$ U' : I' ↦↦$_{C_{\alpha'}}$ $\mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-obj-coprod-axioms*)

mk-ide rf *is-cat-obj-coprod-def*
  |*intro is-cat-obj-coprodI*|
  |*dest is-cat-obj-coprodD*[*dest*]|
  |*elim is-cat-obj-coprodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-coprodD*

Duality.

lemma (**in** *is-cat-obj-prod*) *is-cat-obj-coprod-op*:

op-ntcf $\pi : A >_{CF.\coprod} P : I \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
using *cf-discrete-vdomain-vsubset-Vset*
by (*intro is-cat-obj-coprodI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-obj-prod*) *is-cat-obj-coprod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\pi : A >_{CF.\coprod} P : I \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coprod-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-prod.is-cat-obj-coprod-op'*

lemma (**in** *is-cat-obj-coprod*) *is-cat-obj-prod-op*:
op-ntcf $\pi : U <_{CF.\prod} A : I \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
using *cf-discrete-vdomain-vsubset-Vset*
by (*intro is-cat-obj-prodI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-obj-coprod*) *is-cat-obj-prod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\pi : U <_{CF.\prod} A : I \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-coprod.is-cat-obj-prod-op'*

5.1.2 Universal property

lemma (**in** *is-cat-obj-prod*) *cat-obj-prod-unique-cone'*:
assumes $\pi' : P' <_{CF.cone} :\rightarrow: I\ A\ \mathfrak{C} ::_C I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : P' \mapsto_{\mathfrak{C}} P \wedge (\forall j \in_\circ I.\ \pi'(\lfloor NTMap \rfloor)(\lfloor j \rfloor) = \pi(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{C}} f')$
by
(
rule cat-lim-unique-cone'[
OF assms, *unfolded the-cat-discrete-components*(*1*)
]
)

lemma (**in** *is-cat-obj-prod*) *cat-obj-prod-unique*:
assumes $\pi' : P' <_{CF.\prod} A : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : P' \mapsto_{\mathfrak{C}} P \wedge \pi' = \pi \bullet_{NTCF} ntcf\text{-}const\ (:_C I)\ \mathfrak{C}\ f'$
by (*intro cat-lim-unique*[*OF is-cat-obj-prodD*(*1*)[*OF assms*]])

lemma (**in** *is-cat-obj-prod*) *cat-obj-prod-unique'*:
assumes $\pi' : P' <_{CF.\prod} A : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : P' \mapsto_{\mathfrak{C}} P \wedge (\forall i \in_\circ I.\ \pi'(\lfloor NTMap \rfloor)(\lfloor i \rfloor) = \pi(\lfloor NTMap \rfloor)(\lfloor i \rfloor) \circ_{A\mathfrak{C}} f')$
proof–
interpret π': *is-cat-obj-prod* $\alpha\ I\ A\ \mathfrak{C}\ P'\ \pi'$ **by** (*rule assms*(*1*))
show *?thesis*
by
(
rule cat-lim-unique'[
OF π'*.is-cat-limit-axioms*, *unfolded the-cat-discrete-components*(*1*)

```
        ]
    )
qed

lemma (in is-cat-obj-coprod) cat-obj-coprod-unique-cocone':
  assumes π' : :→: I A 𝔈 >_CF.cocone U' : :_C I ↦↦_Cα 𝔈
  shows ∃!f'. f' : U ↦_𝔈 U' ∧ (∀ j∈∘I. π'(⁅NTMap⁆)(⁅j⁆) = f' ∘_A𝔈 π(⁅NTMap⁆)(⁅j⁆))
  by
    (
      rule cat-colim-unique-cocone'[
        OF assms, unfolded the-cat-discrete-components(1)
        ]
    )

lemma (in is-cat-obj-coprod) cat-obj-coprod-unique:
  assumes π' : A >_CF.∐ U' : I ↦↦_Cα 𝔈
  shows ∃!f'. f' : U ↦_𝔈 U' ∧ π' = ntcf-const (:_C I) 𝔈 f' ·_NTCF π
  by (intro cat-colim-unique[OF is-cat-obj-coprodD(1)[OF assms]])

lemma (in is-cat-obj-coprod) cat-obj-coprod-unique':
  assumes π' : A >_CF.∐ U' : I ↦↦_Cα 𝔈
  shows ∃!f'. f' : U ↦_𝔈 U' ∧ (∀ j∈∘I. π'(⁅NTMap⁆)(⁅j⁆) = f' ∘_A𝔈 π(⁅NTMap⁆)(⁅j⁆))
  by
    (
      rule cat-colim-unique'[
        OF is-cat-obj-coprodD(1)[OF assms], unfolded the-cat-discrete-components
        ]
    )

lemma cat-obj-prod-ex-is-iso-arr:
  assumes π : P <_CF.∏ A : I ↦↦_Cα 𝔈 and π' : P' <_CF.∏ A : I ↦↦_Cα 𝔈
  obtains f where f : P' ↦_iso𝔈 P and π' = π ·_NTCF ntcf-const (:_C I) 𝔈 f
proof–
  interpret π: is-cat-obj-prod α I A 𝔈 P π by (rule assms(1))
  interpret π': is-cat-obj-prod α I A 𝔈 P' π' by (rule assms(2))
  from that show ?thesis
    by
      (
        elim cat-lim-ex-is-iso-arr[
          OF π.is-cat-limit-axioms π'.is-cat-limit-axioms
          ]
      )
qed

lemma cat-obj-prod-ex-is-iso-arr':
  assumes π : P <_CF.∏ A : I ↦↦_Cα 𝔈 and π' : P' <_CF.∏ A : I ↦↦_Cα 𝔈
  obtains f where f : P' ↦_iso𝔈 P
    and ⋀j. j ∈∘ I ⟹ π'(⁅NTMap⁆)(⁅j⁆) = π(⁅NTMap⁆)(⁅j⁆) ∘_A𝔈 f
proof–
  interpret π: is-cat-obj-prod α I A 𝔈 P π by (rule assms(1))
  interpret π': is-cat-obj-prod α I A 𝔈 P' π' by (rule assms(2))
  from that show ?thesis
    by
      (
        elim cat-lim-ex-is-iso-arr'[
          OF π.is-cat-limit-axioms π'.is-cat-limit-axioms,
          unfolded the-cat-discrete-components(1)
          ]
```

)
qed

lemma *cat-obj-coprod-ex-is-iso-arr*:
assumes $\pi : A >_{CF.\coprod} U : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\pi' : A >_{CF.\coprod} U' : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : U \mapsto_{iso\mathfrak{C}} U'$ **and** $\pi' = ntcf\text{-}const\ (:_C\ I)\ \mathfrak{C}\ f \cdot_{NTCF} \pi$
proof–
interpret π: *is-cat-obj-coprod* $\alpha\ I\ A\ \mathfrak{C}\ U\ \pi$ **by** (*rule assms*(*1*))
interpret π': *is-cat-obj-coprod* $\alpha\ I\ A\ \mathfrak{C}\ U'\ \pi'$ **by** (*rule assms*(*2*))
from *that* **show** *?thesis*
by
(
elim cat-colim-ex-is-iso-arr[
OF π.*is-cat-colimit-axioms* π'.*is-cat-colimit-axioms*
]
)
qed

lemma *cat-obj-coprod-ex-is-iso-arr'*:
assumes $\pi : A >_{CF.\coprod} U : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\pi' : A >_{CF.\coprod} U' : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : U \mapsto_{iso\mathfrak{C}} U'$
and $\bigwedge j.\ j \in_\circ I \Longrightarrow \pi'(\lfloor NTMap \rfloor)(\lfloor j \rfloor) = f \circ_{A\mathfrak{C}} \pi(\lfloor NTMap \rfloor)(\lfloor j \rfloor)$
proof–
interpret π: *is-cat-obj-coprod* $\alpha\ I\ A\ \mathfrak{C}\ U\ \pi$ **by** (*rule assms*(*1*))
interpret π': *is-cat-obj-coprod* $\alpha\ I\ A\ \mathfrak{C}\ U'\ \pi'$ **by** (*rule assms*(*2*))
from *that* **show** *?thesis*
by
(
elim cat-colim-ex-is-iso-arr'[
OF π.*is-cat-colimit-axioms* π'.*is-cat-colimit-axioms*,
unfolded the-cat-discrete-components(*1*)
]
)
qed

5.2 Small product and small coproduct

5.2.1 Definition and elementary properties

locale *is-tm-cat-obj-prod* =
is-cat-limit $\alpha\ \langle:_C\ I\rangle\ \mathfrak{C}\ \langle:\mapsto: I\ A\ \mathfrak{C}\rangle\ P\ \pi$ + *tm-cf-discrete* $\alpha\ I\ A\ \mathfrak{C}$
for $\alpha\ I\ A\ \mathfrak{C}\ P\ \pi$

syntax *-is-tm-cat-obj-prod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
$(\langle(\text{-} :/ \text{-} <_{CF.tm.\prod} \text{-} :/ \text{-} \mapsto\mapsto_{C.tm1} \text{-})\rangle\ [51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-tm-cat-obj-prod* $\rightleftharpoons$ *is-tm-cat-obj-prod*
translations $\pi : P <_{CF.tm.\prod} A : I \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \rightleftharpoons$
CONST is-tm-cat-obj-prod $\alpha\ I\ A\ \mathfrak{C}\ P\ \pi$

locale *is-tm-cat-obj-coprod* =
is-cat-colimit $\alpha\ \langle:_C\ I\rangle\ \mathfrak{C}\ \langle:\mapsto: I\ A\ \mathfrak{C}\rangle\ U\ \pi$ + *tm-cf-discrete* $\alpha\ I\ A\ \mathfrak{C}$
for $\alpha\ I\ A\ \mathfrak{C}\ U\ \pi$

syntax *-is-tm-cat-obj-coprod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
$(\langle(\text{-} :/ \text{-} >_{CF.tm.\coprod} \text{-} :/ \text{-} \mapsto\mapsto_{C.tm1} \text{-})\rangle\ [51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-tm-cat-obj-coprod* $\rightleftharpoons$ *is-tm-cat-obj-coprod*
translations $\pi : A >_{CF.tm.\coprod} U : I \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \rightleftharpoons$
CONST is-tm-cat-obj-coprod $\alpha\ I\ A\ \mathfrak{C}\ U\ \pi$

Rules.

lemma (**in** *is-tm-cat-obj-prod*) *is-tm-cat-obj-prod-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF.tm.\prod} A' : I' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-obj-prod-axioms*)

mk-ide rf *is-tm-cat-obj-prod-def*
|*intro is-tm-cat-obj-prodI*|
|*dest is-tm-cat-obj-prodD*[*dest*]|
|*elim is-tm-cat-obj-prodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-obj-prodD*

lemma (**in** *is-tm-cat-obj-coprod*)
is-tm-cat-obj-coprod-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $U' = U$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : A' >_{CF.tm.\coprod} U' : I' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-obj-coprod-axioms*)

mk-ide rf *is-tm-cat-obj-coprod-def*
|*intro is-tm-cat-obj-coprodI*|
|*dest is-tm-cat-obj-coprodD*[*dest*]|
|*elim is-tm-cat-obj-coprodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-obj-coprodD*

Elementary properties.

sublocale *is-tm-cat-obj-prod* ⊆ *is-cat-obj-prod*
by
(
intro is-cat-obj-prodI,
rule is-cat-limit-axioms,
rule cf-discrete-axioms
)

lemmas (**in** *is-tm-cat-obj-prod*) *tm-cat-obj-prod-is-cat-obj-prod* =
is-cat-obj-prod-axioms

sublocale *is-tm-cat-obj-coprod* ⊆ *is-cat-obj-coprod*
by
(
intro is-cat-obj-coprodI,
rule is-cat-colimit-axioms,
rule cf-discrete-axioms
)

lemmas (**in** *is-tm-cat-obj-coprod*) *tm-cat-obj-coprod-is-cat-obj-coprod* =
is-cat-obj-coprod-axioms

sublocale *is-tm-cat-obj-prod* ⊆ *is-tm-cat-limit* α ‹:$_C$ I› $\mathfrak{C}$ ‹:→: I A $\mathfrak{C}$› P π
by
(
intro
is-tm-cat-limitI
is-tm-cat-coneI
is-ntcf-axioms
tm-cf-discrete-the-cf-discrete-is-tm-functor
cat-cone-obj

cat-lim-ua-fo
)

lemmas (**in** *is-tm-cat-obj-prod*) *tm-cat-obj-prod-is-tm-cat-limit* =
is-tm-cat-limit-axioms

sublocale *is-tm-cat-obj-coprod* ⊆ *is-tm-cat-colimit* α ‹:$_C$ I› $\mathfrak{C}$ ‹:→: I A $\mathfrak{C}$› U π
by
(
intro
is-tm-cat-colimitI
is-tm-cat-coconeI
is-ntcf-axioms
tm-cf-discrete-the-cf-discrete-is-tm-functor
cat-cocone-obj
cat-colim-ua-of
)

lemmas (**in** *is-tm-cat-obj-coprod*) *tm-cat-obj-coprod-is-tm-cat-colimit* =
is-tm-cat-colimit-axioms

Duality.

lemma (**in** *is-tm-cat-obj-prod*) *is-tm-cat-obj-coprod-op*:
op-ntcf π : A >$_{CF.tm.\coprod}$ P : I ↦↦$_{C.tm\alpha}$ *op-cat* $\mathfrak{C}$
using *cf-discrete-vdomain-vsubset-Vset*
by (*intro is-tm-cat-obj-coprodI*)
(*cs-concl* **cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)

lemma (**in** *is-tm-cat-obj-prod*) *is-tm-cat-obj-coprod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* π : A >$_{CF.tm.\coprod}$ P : I ↦↦$_{C.tm\alpha}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-obj-coprod-op*)

lemmas [*cat-op-intros*] = *is-tm-cat-obj-prod.is-tm-cat-obj-coprod-op'*

lemma (**in** *is-tm-cat-obj-coprod*) *is-tm-cat-obj-coprod-op*:
op-ntcf π : U <$_{CF.tm.\prod}$ A : I ↦↦$_{C.tm\alpha}$ *op-cat* $\mathfrak{C}$
using *cf-discrete-vdomain-vsubset-Vset*
by (*intro is-tm-cat-obj-prodI*)
(*cs-concl* **cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)

lemma (**in** *is-tm-cat-obj-coprod*) *is-tm-cat-obj-prod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* π : U <$_{CF.tm.\prod}$ A : I ↦↦$_{C.tm\alpha}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-obj-coprod-op*)

lemmas [*cat-op-intros*] = *is-tm-cat-obj-coprod.is-tm-cat-obj-prod-op'*

5.3 Finite product and finite coproduct

locale *is-cat-finite-obj-prod* = *is-cat-obj-prod* α I A $\mathfrak{C}$ P π +
for α I A $\mathfrak{C}$ P π +
assumes *cat-fin-obj-prod-index-in-ω*: I ∈$_\circ$ ω

syntax *-is-cat-finite-obj-prod* :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ *bool*
(‹(- :/ - <$_{CF.\prod.fin}$ - :/ - ↦↦$_{C^1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-finite-obj-prod* ⇌ *is-cat-finite-obj-prod*
translations π : P <$_{CF.\prod.fin}$ A : I ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌

CONST is-cat-finite-obj-prod α I A $\mathfrak{C}$ P π

locale *is-cat-finite-obj-coprod* = *is-cat-obj-coprod* α I A $\mathfrak{C}$ U π
for α I A $\mathfrak{C}$ U π +
assumes *cat-fin-obj-coprod-index-in-ω*: $I \in_\circ \omega$

syntax *-is-cat-finite-obj-coprod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
(‹(- :/ - >$_{CF.\coprod.fin}$ - :/ - ↦↦$_{C1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-finite-obj-coprod* ⇌ *is-cat-finite-obj-coprod*
translations $\pi : A >_{CF.\coprod.fin} U : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$ ⇌
CONST is-cat-finite-obj-coprod α I A $\mathfrak{C}$ U π

lemma (**in** *is-cat-finite-obj-prod*) *cat-fin-obj-prod-index-vfinite*: *vfinite I*
using *cat-fin-obj-prod-index-in-ω* **by** *auto*

sublocale *is-cat-finite-obj-prod* ⊆ *I*: *finite-category* α ‹:$_C$ I›
by (*intro finite-categoryI'*)
(
auto
simp: *NTDom.HomDom.category-axioms the-cat-discrete-components*
intro!: *cat-fin-obj-prod-index-vfinite*
)

lemma (**in** *is-cat-finite-obj-coprod*) *cat-fin-obj-coprod-index-vfinite*:
vfinite I
using *cat-fin-obj-coprod-index-in-ω* **by** *auto*

sublocale *is-cat-finite-obj-coprod* ⊆ *I*: *finite-category* α ‹:$_C$ I›
by (*intro finite-categoryI'*)
(
auto
simp: *NTDom.HomDom.category-axioms the-cat-discrete-components*
intro!: *cat-fin-obj-coprod-index-vfinite*
)

Rules.

lemma (**in** *is-cat-finite-obj-prod*)
is-cat-finite-obj-prod-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF.\prod.fin} A' : I' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-obj-prod-axioms*)

mk-ide rf
is-cat-finite-obj-prod-def[*unfolded is-cat-finite-obj-prod-axioms-def*]
|*intro is-cat-finite-obj-prodI*|
|*dest is-cat-finite-obj-prodD*[*dest*]|
|*elim is-cat-finite-obj-prodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-obj-prodD*

lemma (**in** *is-cat-finite-obj-coprod*)
is-cat-finite-obj-coprod-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $U' = U$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : A' >_{CF.\coprod.fin} U' : I' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-obj-coprod-axioms*)

mk-ide rf
is-cat-finite-obj-coprod-def[*unfolded is-cat-finite-obj-coprod-axioms-def*]

|*intro is-cat-finite-obj-coprodI*|
|*dest is-cat-finite-obj-coprodD*[*dest*]|
|*elim is-cat-finite-obj-coprodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-obj-coprodD*

Duality.

lemma (**in** *is-cat-finite-obj-prod*) *is-cat-finite-obj-coprod-op*:
op-ntcf $\pi : A >_{CF.\coprod.fin} P : I \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-finite-obj-coprodI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps*
cs-intro: *cat-fin-obj-prod-index-in-ω cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-finite-obj-prod*) *is-cat-finite-obj-coprod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\pi : A >_{CF.\coprod.fin} P : I \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-obj-coprod-op*)

lemmas [*cat-op-intros*] = *is-cat-finite-obj-prod.is-cat-finite-obj-coprod-op'*

lemma (**in** *is-cat-finite-obj-coprod*) *is-cat-finite-obj-prod-op*:
op-ntcf $\pi : U <_{CF.\prod.fin} A : I \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-finite-obj-prodI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps*
cs-intro: *cat-fin-obj-coprod-index-in-ω cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-finite-obj-coprod*) *is-cat-finite-obj-prod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\pi : U <_{CF.\prod.fin} A : I \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-obj-prod-op*)

lemmas [*cat-op-intros*] = *is-cat-finite-obj-coprod.is-cat-finite-obj-prod-op'*

5.4 Product and coprodduct of two objects

5.4.1 Definition and elementary properties

locale *is-cat-obj-prod-2* = *is-cat-obj-prod* α ‹$2_{\mathbb{N}}$› ‹*if2 a b*› $\mathfrak{C}$ P π
for α a b $\mathfrak{C}$ P π

syntax *-is-cat-obj-prod-2* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - <$_{CF.\times}$ {-,-} :/ $2_C \mapsto\mapsto_{C1}$ -)› [*51, 51, 51, 51, 51*] *51*)
syntax-consts *-is-cat-obj-prod-2* ⇌ *is-cat-obj-prod-2*
translations $\pi : P <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$ ⇌
CONST is-cat-obj-prod-2 α a b $\mathfrak{C}$ P π

locale *is-cat-obj-coprod-2* = *is-cat-obj-coprod* α ‹$2_{\mathbb{N}}$› ‹*if2 a b*› $\mathfrak{C}$ P π
for α a b $\mathfrak{C}$ P π

syntax *-is-cat-obj-coprod-2* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ {-,-} >$_{CF.\uplus}$ - :/ $2_C \mapsto\mapsto_{C1}$ -)› [*51, 51, 51, 51, 51*] *51*)
syntax-consts *-is-cat-obj-coprod-2* ⇌ *is-cat-obj-coprod-2*

translations $\pi : \{a,b\} >_{CF.\uplus} U : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
CONST is-cat-obj-coprod-2 α *a b* $\mathfrak{C}$ *U* π

abbreviation *proj-fst* **where** *proj-fst* $\pi \equiv$ *vpfst* $(\pi(\!|NTMap|\!))$
abbreviation *proj-snd* **where** *proj-snd* $\pi \equiv$ *vpsnd* $(\pi(\!|NTMap|\!))$

Rules.

lemma (**in** *is-cat-obj-prod-2*) *is-cat-obj-prod-2-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $a' = a$ **and** $b' = b$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF.\times} \{a',b'\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-2-axioms*)

mk-ide rf *is-cat-obj-prod-2-def*
|*intro is-cat-obj-prod-2I*|
|*dest is-cat-obj-prod-2D*[*dest*]|
|*elim is-cat-obj-prod-2E*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-prod-2D*

lemma (**in** *is-cat-obj-coprod-2*) *is-cat-obj-coprod-2-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $a' = a$ **and** $b' = b$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : \{a',b'\} >_{CF.\uplus} P' : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coprod-2-axioms*)

mk-ide rf *is-cat-obj-coprod-2-def*
|*intro is-cat-obj-coprod-2I*|
|*dest is-cat-obj-coprod-2D*[*dest*]|
|*elim is-cat-obj-coprod-2E*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-coprod-2D*

Duality.

lemma (**in** *is-cat-obj-prod-2*) *is-cat-obj-coprod-2-op*:
op-ntcf $\pi : \{a,b\} >_{CF.\uplus} P : 2_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*rule is-cat-obj-coprod-2I*[*OF is-cat-obj-coprod-op*])

lemma (**in** *is-cat-obj-prod-2*) *is-cat-obj-coprod-2-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\pi : \{a,b\} >_{CF.\uplus} P : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coprod-2-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-prod-2.is-cat-obj-coprod-2-op'*

lemma (**in** *is-cat-obj-coprod-2*) *is-cat-obj-prod-2-op*:
op-ntcf $\pi : P <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*rule is-cat-obj-prod-2I*[*OF is-cat-obj-prod-op*])

lemma (**in** *is-cat-obj-coprod-2*) *is-cat-obj-prod-2-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\pi : P <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-2-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-coprod-2.is-cat-obj-prod-2-op'*

Product/coproduct of two objects is a finite product/coproduct.

sublocale *is-cat-obj-prod-2* $\subseteq$ *is-cat-finite-obj-prod* α ‹$2_{\mathbb{N}}$› ‹*if2 a b*› $\mathfrak{C}$ *P* π
proof(*intro is-cat-finite-obj-prodI*)

show $2_{\mathbb{N}} \in_{\circ} \omega$ **by** *simp*
qed (*cs-concl* **cs-shallow cs-simp**: *two*[*symmetric*] **cs-intro**: *cat-lim-cs-intros*)

sublocale *is-cat-obj-coprod-2* $\subseteq$ *is-cat-finite-obj-coprod* α ‹$2_{\mathbb{N}}$› ‹*if2 a b*› $\mathfrak{C}$ P π
proof(*intro is-cat-finite-obj-coprodI*)
show $2_{\mathbb{N}} \in_{\circ} \omega$ **by** *simp*
qed (*cs-concl* **cs-shallow cs-simp**: *two*[*symmetric*] **cs-intro**: *cat-lim-cs-intros*)

Elementary properties.

lemma (**in** *is-cat-obj-prod-2*) *cat-obj-prod-2-lr-in-Obj*:
shows *cat-obj-prod-2-left-in-Obj*[*cat-lim-cs-intros*]: $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and *cat-obj-prod-2-right-in-Obj*[*cat-lim-cs-intros*]: $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
proof–
have *0*: $0 \in_{\circ} 2_{\mathbb{N}}$ **and** *1*: $1_{\mathbb{N}} \in_{\circ} 2_{\mathbb{N}}$ **by** *simp-all*
show $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
by
(
intro
cf-discrete-selector-vrange[*OF 0*, *simplified*]
cf-discrete-selector-vrange[*OF 1*, *simplified*]
)+
qed

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-prod-2.cat-obj-prod-2-lr-in-Obj*

lemma (**in** *is-cat-obj-coprod-2*) *cat-obj-coprod-2-lr-in-Obj*:
shows *cat-obj-coprod-2-left-in-Obj*[*cat-lim-cs-intros*]: $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and *cat-obj-coprod-2-right-in-Obj*[*cat-lim-cs-intros*]: $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
by
(
intro is-cat-obj-prod-2.cat-obj-prod-2-lr-in-Obj[
OF is-cat-obj-prod-2-op, *unfolded cat-op-simps*
]
)+

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-coprod-2.cat-obj-coprod-2-lr-in-Obj*

Utilities/help lemmas.

lemma *helper-I2-proj-fst-proj-snd-iff*:
$(\forall j \in_{\circ} 2_{\mathbb{N}}.\ \pi'(\!|NTMap|\!)(\!|j|\!) = \pi(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{C}} f') \longleftrightarrow$
$(proj\text{-}fst\ \pi' = proj\text{-}fst\ \pi \circ_{A\mathfrak{C}} f' \wedge proj\text{-}snd\ \pi' = proj\text{-}snd\ \pi \circ_{A\mathfrak{C}} f')$
unfolding *two* **by** *auto*

lemma *helper-I2-proj-fst-proj-snd-iff'*:
$(\forall j \in_{\circ} 2_{\mathbb{N}}.\ \pi'(\!|NTMap|\!)(\!|j|\!) = f' \circ_{A\mathfrak{C}} \pi(\!|NTMap|\!)(\!|j|\!)) \longleftrightarrow$
$(proj\text{-}fst\ \pi' = f' \circ_{A\mathfrak{C}} proj\text{-}fst\ \pi \wedge proj\text{-}snd\ \pi' = f' \circ_{A\mathfrak{C}} proj\text{-}snd\ \pi)$
unfolding *two* **by** *auto*

5.4.2 Universal property

lemma (**in** *is-cat-obj-prod-2*) *cat-obj-prod-2-unique-cone'*:
assumes $\pi' : P' <_{CF.cone} :\to: (2_{\mathbb{N}})\ (if2\ a\ b)\ \mathfrak{C} : :_C (2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
$\exists! f'.\ f' : P' \mapsto_{\mathfrak{C}} P \wedge$
$proj\text{-}fst\ \pi' = proj\text{-}fst\ \pi \circ_{A\mathfrak{C}} f' \wedge$
$proj\text{-}snd\ \pi' = proj\text{-}snd\ \pi \circ_{A\mathfrak{C}} f'$
by
(

rule cat-obj-prod-unique-cone'[
OF assms, *unfolded helper-I2-proj-fst-proj-snd-iff*
]
)

lemma (**in** *is-cat-obj-prod-2*) *cat-obj-prod-2-unique*:
assumes $\pi' : P' <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : P' \mapsto_{\mathfrak{C}} P \wedge \pi' = \pi \cdot_{NTCF} \mathit{ntcf\text{-}const}\ (:_C\ (2_{\mathbb{N}}))\ \mathfrak{C}\ f'$
by (*rule cat-obj-prod-unique*[*OF is-cat-obj-prod-2D*[*OF assms*]])

lemma (**in** *is-cat-obj-prod-2*) *cat-obj-prod-2-unique'*:
assumes $\pi' : P' <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
$\exists! f'.\ f' : P' \mapsto_{\mathfrak{C}} P \wedge$
proj-fst π' = *proj-fst* $\pi \circ_{A\mathfrak{C}} f' \wedge$
proj-snd π' = *proj-snd* $\pi \circ_{A\mathfrak{C}} f'$
by
(
rule cat-obj-prod-unique'[
OF is-cat-obj-prod-2D[*OF assms*],
unfolded helper-I2-proj-fst-proj-snd-iff
]
)

lemma (**in** *is-cat-obj-coprod-2*) *cat-obj-coprod-2-unique-cocone'*:
assumes $\pi' : :\!\!\rightarrow : (2_{\mathbb{N}})\ (\mathit{if2}\ a\ b)\ \mathfrak{C} >_{CF.cocone} P' : :_C\ (2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
$\exists! f'.\ f' : P \mapsto_{\mathfrak{C}} P' \wedge$
proj-fst π' = $f' \circ_{A\mathfrak{C}}$ *proj-fst* $\pi \wedge$
proj-snd π' = $f' \circ_{A\mathfrak{C}}$ *proj-snd* π
by
(
rule cat-obj-coprod-unique-cocone'[
OF assms, *unfolded helper-I2-proj-fst-proj-snd-iff'*
]
)

lemma (**in** *is-cat-obj-coprod-2*) *cat-obj-coprod-2-unique*:
assumes $\pi' : \{a,b\} >_{CF.\uplus} P' : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : P \mapsto_{\mathfrak{C}} P' \wedge \pi' = \mathit{ntcf\text{-}const}\ (:_C\ (2_{\mathbb{N}}))\ \mathfrak{C}\ f' \cdot_{NTCF} \pi$
by (*rule cat-obj-coprod-unique*[*OF is-cat-obj-coprod-2D*[*OF assms*]])

lemma (**in** *is-cat-obj-coprod-2*) *cat-obj-coprod-2-unique'*:
assumes $\pi' : \{a,b\} >_{CF.\uplus} P' : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
$\exists! f'.\ f' : P \mapsto_{\mathfrak{C}} P' \wedge$
proj-fst π' = $f' \circ_{A\mathfrak{C}}$ *proj-fst* $\pi \wedge$
proj-snd π' = $f' \circ_{A\mathfrak{C}}$ *proj-snd* π
by
(
rule cat-obj-coprod-unique'[
OF is-cat-obj-coprod-2D[*OF assms*],
unfolded helper-I2-proj-fst-proj-snd-iff'
]
)

lemma *cat-obj-prod-2-ex-is-iso-arr*:
assumes $\pi : P <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$

and $\pi' : P' <_{CF.\times} \{a,b\} : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : P' \mapsto_{iso\mathfrak{C}} P$ **and** $\pi' = \pi \cdot_{NTCF} ntcf\text{-}const\ (:_C\ (2_{\mathbb{N}}))\ \mathfrak{C}\ f$
proof–
interpret π: *is-cat-obj-prod-2* $\alpha\ a\ b\ \mathfrak{C}\ P\ \pi$ **by** (*rule assms*(*1*))
interpret π': *is-cat-obj-prod-2* $\alpha\ a\ b\ \mathfrak{C}\ P'\ \pi'$ **by** (*rule assms*(*2*))
from *that* **show** *?thesis*
by
(
elim cat-obj-prod-ex-is-iso-arr[
OF π.*is-cat-obj-prod-axioms* π'.*is-cat-obj-prod-axioms*
]
)
qed

lemma *cat-obj-coprod-2-ex-is-iso-arr*:
assumes $\pi : \{a,b\} >_{CF.\uplus} U : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\pi' : \{a,b\} >_{CF.\uplus} U' : 2_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : U \mapsto_{iso\mathfrak{C}} U'$ **and** $\pi' = ntcf\text{-}const\ (:_C\ (2_{\mathbb{N}}))\ \mathfrak{C}\ f \cdot_{NTCF} \pi$
proof–
interpret π: *is-cat-obj-coprod-2* $\alpha\ a\ b\ \mathfrak{C}\ U\ \pi$ **by** (*rule assms*(*1*))
interpret π': *is-cat-obj-coprod-2* $\alpha\ a\ b\ \mathfrak{C}\ U'\ \pi'$ **by** (*rule assms*(*2*))
from *that* **show** *?thesis*
by
(
elim cat-obj-coprod-ex-is-iso-arr[
OF π.*is-cat-obj-coprod-axioms* π'.*is-cat-obj-coprod-axioms*
]
)
qed

5.5 Projection cone

5.5.1 Definition and elementary properties

definition *ntcf-obj-prod-base* :: $V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *ntcf-obj-prod-base* $\mathfrak{C}\ I\ F\ P\ f$ =
$[(\lambda j \in_\circ :_C\ I(\!|Obj|\!).\ f\ j),\ cf\text{-}const\ (:_C\ I)\ \mathfrak{C}\ P,\ :\!\rightarrow\!:\ I\ F\ \mathfrak{C},\ :_C\ I,\ \mathfrak{C}]_\circ$

definition *ntcf-obj-coprod-base* :: $V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *ntcf-obj-coprod-base* $\mathfrak{C}\ I\ F\ P\ f$ =
$[(\lambda j \in_\circ :_C\ I(\!|Obj|\!).\ f\ j),\ :\!\rightarrow\!:\ I\ F\ \mathfrak{C},\ cf\text{-}const\ (:_C\ I)\ \mathfrak{C}\ P,\ :_C\ I,\ \mathfrak{C}]_\circ$

Components.

lemma *ntcf-obj-prod-base-components*:
shows *ntcf-obj-prod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTMap|\!) = (\lambda j \in_\circ :_C\ I(\!|Obj|\!).\ f\ j)$
and *ntcf-obj-prod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTDom|\!) = cf\text{-}const\ (:_C\ I)\ \mathfrak{C}\ P$
and *ntcf-obj-prod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTCod|\!) = :\!\rightarrow\!:\ I\ F\ \mathfrak{C}$
and *ntcf-obj-prod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTDGDom|\!) = :_C\ I$
and *ntcf-obj-prod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTDGCod|\!) = \mathfrak{C}$
unfolding *ntcf-obj-prod-base-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

lemma *ntcf-obj-coprod-base-components*:
shows *ntcf-obj-coprod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTMap|\!) = (\lambda j \in_\circ :_C\ I(\!|Obj|\!).\ f\ j)$
and *ntcf-obj-coprod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTDom|\!) = :\!\rightarrow\!:\ I\ F\ \mathfrak{C}$
and *ntcf-obj-coprod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTCod|\!) = cf\text{-}const\ (:_C\ I)\ \mathfrak{C}\ P$
and *ntcf-obj-coprod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTDGDom|\!) = :_C\ I$
and *ntcf-obj-coprod-base* $\mathfrak{C}\ I\ F\ P\ f(\!|NTDGCod|\!) = \mathfrak{C}$

unfolding *ntcf-obj-coprod-base-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Duality.

lemma (**in** *cf-discrete*) *op-ntcf-ntcf-obj-coprod-base*[*cat-op-simps*]:
op-ntcf (*ntcf-obj-coprod-base* $\mathfrak{C}$ *I F P f*) =
ntcf-obj-prod-base (*op-cat* $\mathfrak{C}$) *I F P f*
proof–
note [*cat-op-simps*] = *the-cat-discrete-op*[*OF cf-discrete-vdomain-vsubset-Vset*]
show *?thesis*
unfolding
ntcf-obj-prod-base-def ntcf-obj-coprod-base-def op-ntcf-def nt-field-simps
by (*simp add*: *nat-omega-simps cat-op-simps*)
qed

lemma (**in** *cf-discrete*) *op-ntcf-ntcf-obj-prod-base*[*cat-op-simps*]:
op-ntcf (*ntcf-obj-prod-base* $\mathfrak{C}$ *I F P f*) =
ntcf-obj-coprod-base (*op-cat* $\mathfrak{C}$) *I F P f*
proof–
note [*cat-op-simps*] = *the-cat-discrete-op*[*OF cf-discrete-vdomain-vsubset-Vset*]
show *?thesis*
unfolding
ntcf-obj-prod-base-def ntcf-obj-coprod-base-def op-ntcf-def nt-field-simps
by (*simp add*: *nat-omega-simps cat-op-simps*)
qed

5.5.2 Natural transformation map

mk-VLambda *ntcf-obj-prod-base-components*(*1*)
|*vsv ntcf-obj-prod-base-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-obj-prod-base-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-obj-prod-base-NTMap-app*[*cat-cs-simps*]|

mk-VLambda *ntcf-obj-coprod-base-components*(*1*)
|*vsv ntcf-obj-coprod-base-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-obj-coprod-base-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-obj-coprod-base-NTMap-app*[*cat-cs-simps*]|

5.5.3 Projection natural transformation is a cone

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone*:
assumes $P \in_{\circ} \mathfrak{C}(|Obj|)$ **and** $\bigwedge a.\ a \in_{\circ} I \Longrightarrow f\ a : P \mapsto_{\mathfrak{C}} F\ a$
shows *ntcf-obj-prod-base* $\mathfrak{C}$ *I F P f* : $P <_{CF.cone} :\to: I\ F\ \mathfrak{C} : :_C I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-cat-coneI is-tm-ntcfI' is-ntcfI'*)
from *assms*(*2*) **have** [*cat-cs-intros*]:
$[\![\ a \in_{\circ} I;\ P' = P;\ Fa = F\ a\ ]\!] \Longrightarrow f\ a : P' \mapsto_{\mathfrak{C}} Fa$ **for** *a P' Fa*
by *simp*
show *vfsequence* (*ntcf-obj-prod-base* $\mathfrak{C}$ *I F P f*)
unfolding *ntcf-obj-prod-base-def* **by** *auto*
show *vcard* (*ntcf-obj-prod-base* $\mathfrak{C}$ *I F P f*) = $5_{\mathbb{N}}$
unfolding *ntcf-obj-prod-base-def* **by** (*auto simp*: *nat-omega-simps*)
from *assms* **show** *cf-const* ($:_C I$) $\mathfrak{C}$ $P : :_C I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by
(
cs-concl
cs-intro:
cf-discrete-vdomain-vsubset-Vset
cat-discrete-cs-intros

```
        cat-cs-intros
    )
  show :→: I F 𝔈 : :_C I ↦↦_Cα 𝔈
    by (cs-concl cs-shallow cs-intro: cat-discrete-cs-intros)
  show ntcf-obj-prod-base 𝔈 I F P f(|NTMap|)(|a|) :
    cf-const (:_C I) 𝔈 P(|ObjMap|)(|a|) ↦_𝔈 :→: I F 𝔈(|ObjMap|)(|a|)
    if a ∈_∘ :_C I(|Obj|) for a
  proof–
    from that have a ∈_∘ I unfolding the-cat-discrete-components by simp
    from that this show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-discrete-cs-simps cs-intro: cat-cs-intros
        )
  qed
  show
    ntcf-obj-prod-base 𝔈 I F P f(|NTMap|)(|b|) ∘_A𝔈
      cf-const (:_C I) 𝔈 P(|ArrMap|)(|g|) =
      :→: I F 𝔈(|ArrMap|)(|g|) ∘_A𝔈 ntcf-obj-prod-base 𝔈 I F P f(|NTMap|)(|a|)
    if g : a ↦_:_C I b for a b g
  proof–
    note g = the-cat-discrete-is-arrD[OF that]
    from that g(4)[unfolded g(7–9)] g(1)[unfolded g(7–9)] show ?thesis
      unfolding g(7–9)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-discrete-cs-simps
            cs-intro:
              cf-discrete-vdomain-vsubset-Vset
              cat-cs-intros cat-discrete-cs-intros
        )
  qed
qed
  (
    auto simp:
      assms
      ntcf-obj-prod-base-components
      tm-cf-discrete-the-cf-discrete-is-tm-functor
  )

lemma (in tm-cf-discrete) tm-cf-discrete-ntcf-obj-coprod-base-is-cat-cocone:
  assumes P ∈_∘ 𝔈(|Obj|) and ⋀a. a ∈_∘ I ⟹ f a : F a ↦_𝔈 P
  shows ntcf-obj-coprod-base 𝔈 I F P f :
    :→: I F 𝔈 >_CF.cocone P : :_C I ↦↦_Cα 𝔈
proof–
  note [cat-op-simps] =
    the-cat-discrete-op[OF cf-discrete-vdomain-vsubset-Vset]
    cf-discrete.op-ntcf-ntcf-obj-prod-base[OF cf-discrete-op]
    cf-discrete.cf-discrete-the-cf-discrete-op[OF cf-discrete-op]
  have op-ntcf (ntcf-obj-coprod-base 𝔈 I F P f) :
    P <_CF.cone op-cf (:→: I F 𝔈) : op-cat (:_C I) ↦↦_Cα op-cat 𝔈
    unfolding cat-op-simps
    by
      (
        rule tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone[
          OF tm-cf-discrete-op, unfolded cat-op-simps, OF assms
```

```
        ]
    )
  from is-cat-cone.is-cat-cocone-op[OF this, unfolded cat-op-simps]
  show ?thesis .
qed

lemma (in tm-cf-discrete) tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod:
  assumes P ∈ₒ 𝔈(|Obj|)
    and ⋀a. a ∈ₒ I ⟹ f a : P ↦_𝔈 F a
    and ⋀u' r'.
      [[ u' : r' <_CF.cone :→: I F 𝔈 : :_C I ↦↦_Cα 𝔈 ]] ⟹
        ∃!f'.
          f' : r' ↦_𝔈 P ∧
          u' = ntcf-obj-prod-base 𝔈 I F P f ∙_NTCF ntcf-const (:_C I) 𝔈 f'
  shows ntcf-obj-prod-base 𝔈 I F P f : P <_CF.∏ F : I ↦↦_Cα 𝔈
proof
  (
    intro
      is-cat-obj-prodI
      is-cat-limitI
      tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone[OF assms(1,2), simplified]
      assms(1,3)
  )
  show cf-discrete α I F 𝔈
    by (cs-concl cs-shallow cs-intro: cat-small-discrete-cs-intros)
qed

lemma (in tm-cf-discrete) tm-cf-discrete-ntcf-obj-coprod-base-is-cat-obj-coprod:
  assumes P ∈ₒ 𝔈(|Obj|)
    and ⋀a. a ∈ₒ I ⟹ f a : F a ↦_𝔈 P
    and ⋀u' r'. [[ u' : :→: I F 𝔈 >_CF.cocone r' : :_C I ↦↦_Cα 𝔈 ]] ⟹
      ∃!f'.
        f' : P ↦_𝔈 r' ∧
        u' = ntcf-const (:_C I) 𝔈 f' ∙_NTCF ntcf-obj-coprod-base 𝔈 I F P f
  shows ntcf-obj-coprod-base 𝔈 I F P f : F >_CF.∐ P : I ↦↦_Cα 𝔈
    (is ‹?nc : F >_CF.∐ P : I ↦↦_Cα 𝔈›)
proof–
  let ?np = ‹ntcf-obj-prod-base (op-cat 𝔈) I F P f›
  interpret is-cat-cocone α P ‹:_C I› 𝔈 ‹:→: I F 𝔈› ?nc
    by (intro tm-cf-discrete-ntcf-obj-coprod-base-is-cat-cocone[OF assms(1,2)])
  note [cat-op-simps] =
    the-cat-discrete-op[OF cf-discrete-vdomain-vsubset-Vset]
    cf-discrete.op-ntcf-ntcf-obj-prod-base[OF cf-discrete-op]
    cf-discrete.cf-discrete-the-cf-discrete-op[OF cf-discrete-op]
  have ∃!f'.
    f' : P ↦_𝔈 r ∧
    u = ?np ∙_NTCF ntcf-const (:_C I) (op-cat 𝔈) f'
    if u : r <_CF.cone :→: I F (op-cat 𝔈) : :_C I ↦↦_Cα op-cat 𝔈 for u r
  proof–
    interpret u: is-cat-cone α r ‹:_C I› ‹op-cat 𝔈› ‹:→: I F (op-cat 𝔈)› u
      by (rule that)
    from assms(3)[OF u.is-cat-cocone-op[unfolded cat-op-simps]] obtain g
      where g: g : P ↦_𝔈 r
        and op-u: op-ntcf u = ntcf-const (:_C I) 𝔈 g ∙_NTCF ?nc
        and g-unique:
          [[ g' : P ↦_𝔈 r; op-ntcf u = ntcf-const (:_C I) 𝔈 g' ∙_NTCF ?nc ]] ⟹
            g' = g
      for g'
```

```
      by metis
    show ?thesis
    proof(intro ex1I conjI; (elim conjE)?)
      from op-u have
        op-ntcf (op-ntcf u) = op-ntcf (ntcf-const (:_C I) 𝔈 g ∙_NTCF ?nc)
        by simp
      from this g show u = ?np ∙_NTCF ntcf-const (:_C I) (op-cat 𝔈) g
        by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros)
      fix g' assume prems:
        g' : P ↦_𝔈 r
        u = ?np ∙_NTCF ntcf-const (:_C I) (op-cat 𝔈) g'
      from prems(2) have
        op-ntcf u = op-ntcf (?np ∙_NTCF ntcf-const (:_C I) (op-cat 𝔈) g')
        by simp
      from this prems(1) g have op-ntcf u = ntcf-const (:_C I) 𝔈 g' ∙_NTCF ?nc
        by
          (
            subst (asm)
              the-cat-discrete-op[OF cf-discrete-vdomain-vsubset-Vset, symmetric]
          )
          (
            cs-prems
              cs-simp:
                cat-op-simps
                op-ntcf-ntcf-vcomp[symmetric]
                is-ntcf.ntcf-op-ntcf-op-ntcf
                op-ntcf-ntcf-obj-coprod-base[symmetric]
                op-ntcf-ntcf-const[symmetric]
              cs-intro: cat-cs-intros cat-op-intros
          )
      from g-unique[OF prems(1) this] show g' = g .
    qed (rule g)
  qed
  from is-cat-obj-prod.is-cat-obj-coprod-op
    [
      OF tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod
        [
          OF tm-cf-discrete-op,
          unfolded cat-op-simps,
          OF assms(1,2) this,
          folded op-ntcf-ntcf-obj-coprod-base
        ],
      unfolded cat-op-simps
    ]
  show ?nc : F >_CF.∐ P : I ↦↦_Cα 𝔈.
qed
```

6 Pullbacks and pushouts as limits and colimits

6.1 Pullback and pushout

6.1.1 Definition and elementary properties

The definitions and the elementary properties of the pullbacks and the pushouts can be found, for example, in Chapter III-3 and Chapter III-4 in [9].

locale *is-cat-pullback* =
  is-cat-limit α $\langle \to\bullet\leftarrow_{C} \rangle$ $\mathfrak{C}$ $\langle \langle \mathfrak{a}\to\mathfrak{g}\to\mathfrak{o}\leftarrow\mathfrak{f}\leftarrow\mathfrak{b} \rangle_{CF\mathfrak{C}} \rangle$ X x +
  cf-scospan α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$
  for α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ X x

syntax *-is-cat-pullback* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  $(\langle (\text{-} :/ \text{-} <_{CF.pb} \text{-}\to\text{-}\to\text{-}\leftarrow\text{-}\leftarrow\text{-} \mapsto\mapsto_{C1} \text{-}) \rangle$ [*51*, *51*, *51*, *51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-pullback* $\rightleftharpoons$ *is-cat-pullback*
translations $x : X <_{CF.pb} \mathfrak{a}\to\mathfrak{g}\to\mathfrak{o}\leftarrow\mathfrak{f}\leftarrow\mathfrak{b} \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-cat-pullback α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ X x

locale *is-cat-pushout* =
  is-cat-colimit α $\langle \leftarrow\bullet\to_{C} \rangle$ $\mathfrak{C}$ $\langle \langle \mathfrak{a}\leftarrow\mathfrak{g}\leftarrow\mathfrak{o}\to\mathfrak{f}\to\mathfrak{b} \rangle_{CF\mathfrak{C}} \rangle$ X x +
  cf-sspan α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$
  for α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ X x

syntax *-is-cat-pushout* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  $(\langle (\text{-} :/ \text{-}\leftarrow\text{-}\leftarrow\text{-}\to\text{-}\to\text{-} >_{CF.po} \text{-} \mapsto\mapsto_{C1} \text{-}) \rangle$ [*51*, *51*, *51*, *51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-pushout* $\rightleftharpoons$ *is-cat-pushout*
translations $x : \mathfrak{a}\leftarrow\mathfrak{g}\leftarrow\mathfrak{o}\to\mathfrak{f}\to\mathfrak{b} >_{CF.po} X \mapsto\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-cat-pushout α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ X x

Rules.

lemma (**in** *is-cat-pullback*) *is-cat-pullback-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$
    and $\mathfrak{a}' = \mathfrak{a}$
    and $\mathfrak{g}' = \mathfrak{g}$
    and $\mathfrak{o}' = \mathfrak{o}$
    and $\mathfrak{f}' = \mathfrak{f}$
    and $\mathfrak{b}' = \mathfrak{b}$
    and $\mathfrak{C}' = \mathfrak{C}$
    and $X' = X$
  shows $x : X' <_{CF.pb} \mathfrak{a}'\to\mathfrak{g}'\to\mathfrak{o}'\leftarrow\mathfrak{f}'\leftarrow\mathfrak{b}' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-pullback-axioms*)

mk-ide rf *is-cat-pullback-def*
  |*intro is-cat-pullbackI*|
  |*dest is-cat-pullbackD*[*dest*]|
  |*elim is-cat-pullbackE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-pullbackD*

lemma (**in** *is-cat-pushout*) *is-cat-pushout-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$
    and $\mathfrak{a}' = \mathfrak{a}$
    and $\mathfrak{g}' = \mathfrak{g}$
    and $\mathfrak{o}' = \mathfrak{o}$
    and $\mathfrak{f}' = \mathfrak{f}$
    and $\mathfrak{b}' = \mathfrak{b}$
    and $\mathfrak{C}' = \mathfrak{C}$

and $X' = X$
shows $x : \mathfrak{a}'\leftarrow\mathfrak{g}'\leftarrow\mathfrak{o}'\rightarrow\mathfrak{f}'\rightarrow\mathfrak{b}' >_{CF.po} X' \mapsto\mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-pushout-axioms*)

mk-ide rf *is-cat-pushout-def*
|*intro is-cat-pushoutI*|
|*dest is-cat-pushoutD*[*dest*]|
|*elim is-cat-pushoutE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-pushoutD*

Duality.

lemma (**in** *is-cat-pullback*) *is-cat-pushout-op*:
op-ntcf x : $\mathfrak{a}\leftarrow\mathfrak{g}\leftarrow\mathfrak{o}\rightarrow\mathfrak{f}\rightarrow\mathfrak{b} >_{CF.po} X \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-pushoutI*)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)+

lemma (**in** *is-cat-pullback*) *is-cat-pushout-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf x* : $\mathfrak{a}\leftarrow\mathfrak{g}\leftarrow\mathfrak{o}\rightarrow\mathfrak{f}\rightarrow\mathfrak{b} >_{CF.po} X \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-pushout-op*)

lemmas [*cat-op-intros*] = *is-cat-pullback.is-cat-pushout-op'*

lemma (**in** *is-cat-pushout*) *is-cat-pullback-op*:
op-ntcf x : $X <_{CF.pb} \mathfrak{a}\rightarrow\mathfrak{g}\rightarrow\mathfrak{o}\leftarrow\mathfrak{f}\leftarrow\mathfrak{b} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-pullbackI*)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)+

lemma (**in** *is-cat-pushout*) *is-cat-pullback-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf x* : $X <_{CF.pb} \mathfrak{a}\rightarrow\mathfrak{g}\rightarrow\mathfrak{o}\leftarrow\mathfrak{f}\leftarrow\mathfrak{b} \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-pullback-op*)

lemmas [*cat-op-intros*] = *is-cat-pushout.is-cat-pullback-op'*

Elementary properties.

lemma *cat-cone-cospan*:
assumes $x : X <_{CF.cone} \langle\mathfrak{a}\rightarrow\mathfrak{g}\rightarrow\mathfrak{o}\leftarrow\mathfrak{f}\leftarrow\mathfrak{b}\rangle_{CF\mathfrak{C}} : \rightarrow\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and *cf-scospan* α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$
shows $x(\!|NTMap|\!)(\!|\mathfrak{o}_{SS}|\!) = \mathfrak{g} \circ_{A\mathfrak{C}} x(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!)$
and $x(\!|NTMap|\!)(\!|\mathfrak{o}_{SS}|\!) = \mathfrak{f} \circ_{A\mathfrak{C}} x(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!)$
and $\mathfrak{g} \circ_{A\mathfrak{C}} x(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) = \mathfrak{f} \circ_{A\mathfrak{C}} x(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!)$
proof–
interpret *x*: *is-cat-cone* α X $\langle\rightarrow\bullet\leftarrow_C\rangle$ $\mathfrak{C}$ $\langle\langle\mathfrak{a}\rightarrow\mathfrak{g}\rightarrow\mathfrak{o}\leftarrow\mathfrak{f}\leftarrow\mathfrak{b}\rangle_{CF\mathfrak{C}}\rangle$ x
by (*rule assms*(*1*))
interpret *cospan*: *cf-scospan* α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ **by** (*rule assms*(*2*))
have $\mathfrak{g}_{SS}$: $\mathfrak{g}_{SS} : \mathfrak{a}_{SS} \mapsto_{\rightarrow\bullet\leftarrow_C} \mathfrak{o}_{SS}$ **and** $\mathfrak{f}_{SS}$: $\mathfrak{f}_{SS} : \mathfrak{b}_{SS} \mapsto_{\rightarrow\bullet\leftarrow_C} \mathfrak{o}_{SS}$
by (*cs-concl* **cs-intro**: *cat-ss-cs-intros*)+
note *x.cat-cone-Comp-commute*[*cat-cs-simps del*]
from *x.ntcf-Comp-commute*[*OF* $\mathfrak{g}_{SS}$] $\mathfrak{g}_{SS}$ $\mathfrak{f}_{SS}$ **show**
$x(\!|NTMap|\!)(\!|\mathfrak{o}_{SS}|\!) = \mathfrak{g} \circ_{A\mathfrak{C}} x(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!)$
by
(
cs-prems **cs-shallow**
cs-simp: *cat-ss-cs-simps cat-cs-simps* **cs-intro**: *cat-cs-intros*
)
moreover from *x.ntcf-Comp-commute*[*OF* $\mathfrak{f}_{SS}$] $\mathfrak{g}_{SS}$ $\mathfrak{f}_{SS}$ **show**

```
    x(|NTMap|)(|𝔬_SS|) = 𝔣 ∘_A𝔈 x(|NTMap|)(|𝔟_SS|)
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-ss-cs-simps cat-cs-simps cs-intro: cat-cs-intros
      )
  ultimately show 𝔤 ∘_A𝔈 x(|NTMap|)(|𝔞_SS|) = 𝔣 ∘_A𝔈 x(|NTMap|)(|𝔟_SS|) by simp
qed

lemma (in is-cat-pullback) cat-pb-cone-cospan:
  shows x(|NTMap|)(|𝔬_SS|) = 𝔤 ∘_A𝔈 x(|NTMap|)(|𝔞_SS|)
    and x(|NTMap|)(|𝔬_SS|) = 𝔣 ∘_A𝔈 x(|NTMap|)(|𝔟_SS|)
    and 𝔤 ∘_A𝔈 x(|NTMap|)(|𝔞_SS|) = 𝔣 ∘_A𝔈 x(|NTMap|)(|𝔟_SS|)
  by (all‹rule cat-cone-cospan[OF is-cat-cone-axioms cf-scospan-axioms]›)

lemma cat-cocone-span:
  assumes x : ⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_CF𝔈 >_CF.cocone X : ←•→_C ↦↦_C𝔄 𝔈
    and cf-sspan 𝔄 𝔞 𝔤 𝔬 𝔣 𝔟 𝔈
  shows x(|NTMap|)(|𝔬_SS|) = x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 𝔤
    and x(|NTMap|)(|𝔬_SS|) = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 𝔣
    and x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 𝔤 = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 𝔣
proof–
  interpret x: is-cat-cocone 𝔄 X ‹←•→_C› 𝔈 ‹⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_CF𝔈› x
    by (rule assms(1))
  interpret span: cf-sspan 𝔄 𝔞 𝔤 𝔬 𝔣 𝔟 𝔈 by (rule assms(2))
  note op =
    cat-cone-cospan
      [
        OF
          x.is-cat-cone-op[unfolded cat-op-simps]
          span.cf-scospan-op,
          unfolded cat-op-simps
      ]
  from op(1) show x(|NTMap|)(|𝔬_SS|) = x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 𝔤
    by
      (
        cs-prems
          cs-simp: cat-ss-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-ss-cs-intros
      )
  moreover from op(2) show x(|NTMap|)(|𝔬_SS|) = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 𝔣
    by
      (
        cs-prems
          cs-simp: cat-ss-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-ss-cs-intros
      )
  ultimately show x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 𝔤 = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 𝔣 by auto
qed

lemma (in is-cat-pushout) cat-po-cocone-span:
  shows x(|NTMap|)(|𝔬_SS|) = x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 𝔤
    and x(|NTMap|)(|𝔬_SS|) = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 𝔣
    and x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 𝔤 = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 𝔣
  by (all‹rule cat-cocone-span[OF is-cat-cocone-axioms cf-sspan-axioms]›)
```

6.1.2 Universal property

lemma *is-cat-pullbackI'*:
  assumes $x : X <_{CF.cone} \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \to\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and *cf-scospan* $\alpha\ \mathfrak{a}\ \mathfrak{g}\ \mathfrak{o}\ \mathfrak{f}\ \mathfrak{b}\ \mathfrak{C}$
    and $\bigwedge x'\ X'.\ x' : X' <_{CF.cone} \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \to\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C} \Longrightarrow$
      $\exists! f'.$
        $f' : X' \mapsto_{\mathfrak{C}} X \wedge$
        $x'(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) = x(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) \circ_{A\mathfrak{C}} f' \wedge$
        $x'(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!) = x(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!) \circ_{A\mathfrak{C}} f'$
  shows $x : X <_{CF.pb} \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-cat-pullbackI is-cat-limitI*)

  show $x : X <_{CF.cone} \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \to\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by (*rule assms*(*1*))
  interpret *x*: *is-cat-cone* $\alpha\ X\ \langle \to\bullet\leftarrow_C \rangle\ \mathfrak{C}\ \langle \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} \rangle\ x$
    by (*rule assms*(*1*))
  show *cf-scospan* $\alpha\ \mathfrak{a}\ \mathfrak{g}\ \mathfrak{o}\ \mathfrak{f}\ \mathfrak{b}\ \mathfrak{C}$ **by** (*rule assms*(*2*))
  interpret *cospan*: *cf-scospan* $\alpha\ \mathfrak{a}\ \mathfrak{g}\ \mathfrak{o}\ \mathfrak{f}\ \mathfrak{b}\ \mathfrak{C}$ **by** (*rule assms*(*2*))

  fix $u'\ r'$ **assume** *prems*:
    $u' : r' <_{CF.cone} \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \to\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$

  interpret *u'*: *is-cat-cone* $\alpha\ r'\ \langle \to\bullet\leftarrow_C \rangle\ \mathfrak{C}\ \langle \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} \rangle\ u'$
    by (*rule prems*)

  from *assms*(*3*)[*OF prems*] **obtain** f'
    where *f'*: $f' : r' \mapsto_{\mathfrak{C}} X$
      and *u'-𝔞SS*: $u'(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) = x(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) \circ_{A\mathfrak{C}} f'$
      and *u'-𝔟SS*: $u'(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!) = x(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!) \circ_{A\mathfrak{C}} f'$
      and *unique-f'*: $\bigwedge f''.$
        $[\![$
          $f'' : r' \mapsto_{\mathfrak{C}} X;$
          $u'(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) = x(\!|NTMap|\!)(\!|\mathfrak{a}_{SS}|\!) \circ_{A\mathfrak{C}} f'';$
          $u'(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!) = x(\!|NTMap|\!)(\!|\mathfrak{b}_{SS}|\!) \circ_{A\mathfrak{C}} f''$
        $]\!] \Longrightarrow f'' = f'$
    by *metis*

  show $\exists! f'.\ f' : r' \mapsto_{\mathfrak{C}} X \wedge u' = x \cdot_{NTCF} ntcf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ f'$
  proof(*intro ex1I conjI*; (*elim conjE*)*?*)

    show $u' = x \cdot_{NTCF} ntcf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ f'$
    proof(*rule ntcf-eqI*)
      show $u' : cf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ r' \mapsto_{CF} \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \to\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
        by (*rule u'.is-ntcf-axioms*)
      from f' **show**
        $x \cdot_{NTCF} ntcf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ f' :$
          $cf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ r' \mapsto_{CF} \langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} :$
          $\to\bullet\leftarrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
        by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
      from f' **have** *dom-rhs*:
        $\mathcal{D}_\circ\ ((x \cdot_{NTCF} ntcf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ f')(\!|NTMap|\!)) = \to\bullet\leftarrow_C(\!|Obj|\!)$
        by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
      show $u'(\!|NTMap|\!) = (x \cdot_{NTCF} ntcf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ f')(\!|NTMap|\!)$
      proof(*rule vsv-eqI*, *unfold cat-cs-simps dom-rhs*)
        fix *a* **assume** *prems'*: $a \in_\circ \to\bullet\leftarrow_C(\!|Obj|\!)$
        from *this f' x.is-ntcf-axioms* **show**
          $u'(\!|NTMap|\!)(\!|a|\!) = (x \cdot_{NTCF} ntcf\text{-}const \to\bullet\leftarrow_C \mathfrak{C}\ f')(\!|NTMap|\!)(\!|a|\!)$

```
        by (elim the-cat-scospan-ObjE; simp only:)
          (
            cs-concl
              cs-simp:
                cat-cs-simps cat-ss-cs-simps
                u'-𝔟_SS u'-𝔞_SS
                cat-cone-cospan(1)[OF assms(1,2)]
                cat-cone-cospan(1)[OF prems assms(2)]
              cs-intro: cat-cs-intros cat-ss-cs-intros
          )+
      qed (cs-concl cs-intro: cat-cs-intros | auto)+
    qed simp-all

    fix f'' assume prems:
      f'' : r' ↦_ℭ X u' = x ⋆_NTCF ntcf-const →•←_C ℭ f''
    have 𝔞_SS: 𝔞_SS ∈_∘ →•←_C(|Obj|) and 𝔟_SS: 𝔟_SS ∈_∘ →•←_C(|Obj|)
      by (cs-concl cs-intro: cat-ss-cs-intros)+
    have u'(|NTMap|)(|a|) = x(|NTMap|)(|a|) ∘_Aℭ f'' if a ∈_∘ →•←_C(|Obj|) for a
    proof–
      from prems(2) have
        u'(|NTMap|)(|a|) = (x ⋆_NTCF ntcf-const →•←_C ℭ f'')(|NTMap|)(|a|)
        by simp
      from this that prems(1) show u'(|NTMap|)(|a|) = x(|NTMap|)(|a|) ∘_Aℭ f''
        by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed
    from unique-f'[OF prems(1) this[OF 𝔞_SS] this[OF 𝔟_SS]] show f'' = f'.

  qed (intro f')

qed

lemma is-cat-pushoutI':
  assumes x : ⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_CFℭ >_CF.cocone X : ←•→_C ↦↦_Cα ℭ
    and cf-sspan α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ
    and ⋀x' X'. x' : ⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_CFℭ >_CF.cocone X' : ←•→_C ↦↦_Cα ℭ ⟹
      ∃!f'.
        f' : X ↦_ℭ X' ∧
        x'(|NTMap|)(|𝔞_SS|) = f' ∘_Aℭ x(|NTMap|)(|𝔞_SS|) ∧
        x'(|NTMap|)(|𝔟_SS|) = f' ∘_Aℭ x(|NTMap|)(|𝔟_SS|)
  shows x : 𝔞←𝔤←𝔬→𝔣→𝔟 >_CF.po X ↦↦_Cα ℭ
proof–
  interpret x: is-cat-cocone α X ‹←•→_C› ℭ ‹⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_CFℭ› x
    by (rule assms(1))
  interpret span: cf-sspan α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ by (rule assms(2))
  have assms-3':
    ∃!f'.
      f' : X ↦_ℭ X' ∧
      x'(|NTMap|)(|𝔞_SS|) = x(|NTMap|)(|𝔞_SS|) ∘_A op-cat ℭ f' ∧
      x'(|NTMap|)(|𝔟_SS|) = x(|NTMap|)(|𝔟_SS|) ∘_A op-cat ℭ f'
    if x' : X' <_CF.cone ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_CF op-cat ℭ : →•←_C ↦↦_Cα op-cat ℭ
    for x' X'
  proof–
    from that(1) have [cat-op-simps]:
      f' : X ↦_ℭ X' ∧
      x'(|NTMap|)(|𝔞_SS|) = x(|NTMap|)(|𝔞_SS|) ∘_A op-cat ℭ f' ∧
      x'(|NTMap|)(|𝔟_SS|) = x(|NTMap|)(|𝔟_SS|) ∘_A op-cat ℭ f' ⟷
        f' : X ↦_ℭ X' ∧
        x'(|NTMap|)(|𝔞_SS|) = f' ∘_Aℭ x(|NTMap|)(|𝔞_SS|) ∧
```

```
    x'(|NTMap|)(|𝔟_SS|) = f' ∘_A𝔈 x(|NTMap|)(|𝔟_SS|)
  for f'
  by (intro iffI conjI; (elim conjE)?)
   (
    cs-concl
     cs-simp: category.op-cat-Comp[symmetric] cat-op-simps cat-cs-simps
     cs-intro: cat-cs-intros cat-ss-cs-intros
   )+
 interpret x':
  is-cat-cone α X' ‹→•←_C› ‹op-cat 𝔈› ‹⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_CF op-cat 𝔈› x'
  by (rule that)
 show ?thesis
  unfolding cat-op-simps
  by
   (
    rule assms(3)[
     OF x'.is-cat-cocone-op[unfolded cat-op-simps],
     unfolded cat-op-simps
     ]
   )
qed
interpret op-x: is-cat-pullback α 𝔞 𝔤 𝔬 𝔣 𝔟 ‹op-cat 𝔈› X ‹op-ntcf x›
 using
  is-cat-pullbackI'
   [
    OF x.is-cat-cone-op[unfolded cat-op-simps]
    span.cf-scospan-op,
    unfolded cat-op-simps,
    OF assms-3'
   ]
 by simp
show x : 𝔞←𝔤←𝔬→𝔣→𝔟 >_CF.po X ↦↦_Cα 𝔈
 by (rule op-x.is-cat-pushout-op[unfolded cat-op-simps])
qed

lemma (in is-cat-pullback) cat-pb-unique-cone:
 assumes x' : X' <_CF.cone ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_CF𝔈 : →•←_C ↦↦_Cα 𝔈
 shows ∃!f'.
  f' : X' ↦_𝔈 X ∧
  x'(|NTMap|)(|𝔞_SS|) = x(|NTMap|)(|𝔞_SS|) ∘_A𝔈 f' ∧
  x'(|NTMap|)(|𝔟_SS|) = x(|NTMap|)(|𝔟_SS|) ∘_A𝔈 f'
proof–
 interpret x': is-cat-cone α X' ‹→•←_C› 𝔈 ‹⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_CF𝔈› x'
  by (rule assms)
 from cat-lim-ua-fo[OF assms] obtain f'
  where f': f' : X' ↦_𝔈 X
   and x'-def: x' = x •_NTCF ntcf-const →•←_C 𝔈 f'
   and unique-f': ⋀f''.
    [[ f'' : X' ↦_𝔈 X; x' = x •_NTCF ntcf-const →•←_C 𝔈 f'' ]] ⟹
    f'' = f'
  by auto
 have 𝔞_SS: 𝔞_SS ∈_∘ →•←_C(|Obj|) and 𝔟_SS: 𝔟_SS ∈_∘ →•←_C(|Obj|)
  by (cs-concl cs-intro: cat-ss-cs-intros)+
 show ?thesis
 proof(intro ex1I conjI; (elim conjE)?)
  show f' : X' ↦_𝔈 X by (rule f')
  have x'(|NTMap|)(|a|) = x(|NTMap|)(|a|) ∘_A𝔈 f' if a ∈_∘ →•←_C(|Obj|) for a
  proof–
```

```
    from x'-def have
      x'(|NTMap|)(|a|) = (x ·NTCF ntcf-const →·←C 𝔈 f')(|NTMap|)(|a|)
      by simp
    from this that f' show x'(|NTMap|)(|a|) = x(|NTMap|)(|a|) ∘A𝔈 f'
      by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed
  from this[OF 𝔞SS] this[OF 𝔟SS] show
    x'(|NTMap|)(|𝔞SS|) = x(|NTMap|)(|𝔞SS|) ∘A𝔈 f'
    x'(|NTMap|)(|𝔟SS|) = x(|NTMap|)(|𝔟SS|) ∘A𝔈 f'
    by auto
  fix f'' assume prems':
    f'' : X' ↦𝔈 X
    x'(|NTMap|)(|𝔞SS|) = x(|NTMap|)(|𝔞SS|) ∘A𝔈 f''
    x'(|NTMap|)(|𝔟SS|) = x(|NTMap|)(|𝔟SS|) ∘A𝔈 f''
  have x' = x ·NTCF ntcf-const →·←C 𝔈 f''
  proof(rule ntcf-eqI)
    show x' : cf-const →·←C 𝔈 X' ↦CF ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩CF𝔈 : →·←C ↦↦Cα 𝔈
      by (rule x'.is-ntcf-axioms)
    from prems'(1) show
      x ·NTCF ntcf-const →·←C 𝔈 f'' :
        cf-const →·←C 𝔈 X' ↦CF ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩CF𝔈 :
        →·←C ↦↦Cα 𝔈
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    have dom-lhs: 𝒟∘ (x'(|NTMap|)) = →·←C(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from prems'(1) have dom-rhs:
      𝒟∘ ((x ·NTCF ntcf-const →·←C 𝔈 f'')(|NTMap|)) = →·←C(|Obj|)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show x'(|NTMap|) = (x ·NTCF ntcf-const →·←C 𝔈 f'')(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume prems'': a ∈∘ →·←C(|Obj|)
      from this prems'(1) show
        x'(|NTMap|)(|a|) = (x ·NTCF ntcf-const →·←C 𝔈 f'')(|NTMap|)(|a|)
        by (elim the-cat-scospan-ObjE; simp only:)
          (
            cs-concl
              cs-simp:
                prems'(2,3)
                cat-cone-cospan(1,2)[OF assms cf-scospan-axioms]
                cat-pb-cone-cospan
                cat-ss-cs-simps cat-cs-simps
              cs-intro: cat-ss-cs-intros cat-cs-intros
          )+
    qed (auto simp: cat-cs-intros)
  qed simp-all
  from unique-f'[OF prems'(1) this] show f'' = f'.
 qed
qed

lemma (in is-cat-pullback) cat-pb-unique:
 assumes x' : X' <CF.pb 𝔞→𝔤→𝔬←𝔣←𝔟 ↦↦Cα 𝔈
 shows ∃!f'. f' : X' ↦𝔈 X ∧ x' = x ·NTCF ntcf-const →·←C 𝔈 f'
 by (rule cat-lim-unique[OF is-cat-pullbackD(1)[OF assms]])

lemma (in is-cat-pullback) cat-pb-unique':
 assumes x' : X' <CF.pb 𝔞→𝔤→𝔬←𝔣←𝔟 ↦↦Cα 𝔈
 shows ∃!f'.
   f' : X' ↦𝔈 X ∧
```

$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) = x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) \circ_{A\mathfrak{C}} f' \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) = x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) \circ_{A\mathfrak{C}} f'$
proof–
interpret x': *is-cat-pullback* α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ X' x' **by** (*rule assms*(*1*))
show *?thesis* **by** (*rule cat-pb-unique-cone*[*OF x'.is-cat-cone-axioms*])
qed

lemma (**in** *is-cat-pushout*) *cat-po-unique-cocone*:
assumes $x' : \langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} >_{CF.cocone} X' : \leftarrow\!\bullet\!\rightarrow_C \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'$.
$f' : X \mapsto_{\mathfrak{C}} X' \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) = f' \circ_{A\mathfrak{C}} x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) = f' \circ_{A\mathfrak{C}} x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor)$
proof–
interpret x': *is-cat-cocone* α X' $\langle \leftarrow\!\bullet\!\rightarrow_C \rangle$ $\mathfrak{C}$ $\langle \langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} \rangle$ x'
by (*rule assms*(*1*))
have [*cat-op-simps*]:
$f' : X \mapsto_{\mathfrak{C}} X' \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) = x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) \circ_{A\, op\text{-}cat\ \mathfrak{C}} f' \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) = x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) \circ_{A\, op\text{-}cat\ \mathfrak{C}} f' \longleftrightarrow$
$f' : X \mapsto_{\mathfrak{C}} X' \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) = f' \circ_{A\mathfrak{C}} x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) = f' \circ_{A\mathfrak{C}} x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor)$
for f'
by (*intro iffI conjI*; (*elim conjE*) *?*)
(
cs-concl
cs-simp: *category.op-cat-Comp*[*symmetric*] *cat-op-simps cat-cs-simps*
cs-intro: *cat-cs-intros cat-ss-cs-intros*
)+
show *?thesis*
by
(
rule is-cat-pullback.cat-pb-unique-cone[
OF is-cat-pullback-op x'.is-cat-cone-op[*unfolded cat-op-simps*],
unfolded cat-op-simps
]
)
qed

lemma (**in** *is-cat-pushout*) *cat-po-unique*:
assumes $x' : \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} >_{CF.po} X' \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'$. $f' : X \mapsto_{\mathfrak{C}} X' \wedge x' = ntcf\text{-}const \leftarrow\!\bullet\!\rightarrow_C \mathfrak{C}\ f' \bullet_{NTCF} x$
by (*rule cat-colim-unique*[*OF is-cat-pushoutD*(*1*)[*OF assms*]])

lemma (**in** *is-cat-pushout*) *cat-po-unique'*:
assumes $x' : \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} >_{CF.po} X' \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'$.
$f' : X \mapsto_{\mathfrak{C}} X' \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) = f' \circ_{A\mathfrak{C}} x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{SS} \rfloor) \wedge$
$x'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor) = f' \circ_{A\mathfrak{C}} x(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{SS} \rfloor)$
proof–
interpret x': *is-cat-pushout* α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ X' x' **by** (*rule assms*(*1*))
show *?thesis* **by** (*rule cat-po-unique-cocone*[*OF x'.is-cat-cocone-axioms*])
qed

lemma *cat-pullback-ex-is-iso-arr*:
assumes $x : X <_{CF.pb} \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$

```
    and x' : X' <_{CF.pb} 𝔞→𝔤→𝔬←𝔣←𝔟 ↦↦_{Cα} ℭ
  obtains f where f : X' ↦_{iso ℭ} X
    and x' = x ⋆_{NTCF} ntcf-const →•← _C  ℭ f
proof–
  interpret x: is-cat-pullback α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X x by (rule assms(1))
  interpret x': is-cat-pullback α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X' x' by (rule assms(2))
  from that show ?thesis
    by
      (
        elim cat-lim-ex-is-iso-arr[
          OF x.is-cat-limit-axioms x'.is-cat-limit-axioms
          ]
      )
qed

lemma cat-pullback-ex-is-iso-arr':
  assumes x : X <_{CF.pb} 𝔞→𝔤→𝔬←𝔣←𝔟 ↦↦_{Cα} ℭ
    and x' : X' <_{CF.pb} 𝔞→𝔤→𝔬←𝔣←𝔟 ↦↦_{Cα} ℭ
  obtains f where f : X' ↦_{iso ℭ} X
    and x'(|NTMap|)(|𝔞_{SS}|) = x(|NTMap|)(|𝔞_{SS}|) ∘_{Aℭ} f
    and x'(|NTMap|)(|𝔟_{SS}|) = x(|NTMap|)(|𝔟_{SS}|) ∘_{Aℭ} f
proof–
  interpret x: is-cat-pullback α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X x by (rule assms(1))
  interpret x': is-cat-pullback α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X' x' by (rule assms(2))
  obtain f where f: f : X' ↦_{iso ℭ} X
    and j ∈_∘ →•←_C(|Obj|) ⟹ x'(|NTMap|)(|j|) = x(|NTMap|)(|j|) ∘_{Aℭ} f for j
    by
      (
        elim cat-lim-ex-is-iso-arr'[
          OF x.is-cat-limit-axioms x'.is-cat-limit-axioms
          ]
      )
  then have
    x'(|NTMap|)(|𝔞_{SS}|) = x(|NTMap|)(|𝔞_{SS}|) ∘_{Aℭ} f
    x'(|NTMap|)(|𝔟_{SS}|) = x(|NTMap|)(|𝔟_{SS}|) ∘_{Aℭ} f
    by (auto simp: cat-ss-cs-intros)
  with f show ?thesis using that by simp
qed

lemma cat-pushout-ex-is-iso-arr:
  assumes x : 𝔞←𝔤←𝔬→𝔣→𝔟 >_{CF.po} X ↦↦_{Cα} ℭ
    and x' : 𝔞←𝔤←𝔬→𝔣→𝔟 >_{CF.po} X' ↦↦_{Cα} ℭ
  obtains f where f : X ↦_{iso ℭ} X'
    and x' = ntcf-const ←•→_C ℭ f ⋆_{NTCF} x
proof–
  interpret x: is-cat-pushout α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X x by (rule assms(1))
  interpret x': is-cat-pushout α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X' x' by (rule assms(2))
  from that show ?thesis
    by
      (
        elim cat-colim-ex-is-iso-arr[
          OF x.is-cat-colimit-axioms x'.is-cat-colimit-axioms
          ]
      )
qed

lemma cat-pushout-ex-is-iso-arr':
  assumes x : 𝔞←𝔤←𝔬→𝔣→𝔟 >_{CF.po} X ↦↦_{Cα} ℭ
```

```
    and x' : 𝔞←𝔤←𝔬→𝔣→𝔟 >_{CF.po} X' ↦↦_{Cα} ℭ
  obtains f where f : X ↦_{iso ℭ} X'
    and x'(⦇NTMap⦈)(⦇𝔞_{SS}⦈) = f ∘_{A ℭ} x(⦇NTMap⦈)(⦇𝔞_{SS}⦈)
    and x'(⦇NTMap⦈)(⦇𝔟_{SS}⦈) = f ∘_{A ℭ} x(⦇NTMap⦈)(⦇𝔟_{SS}⦈)
proof–
  interpret x: is-cat-pushout α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X x by (rule assms(1))
  interpret x': is-cat-pushout α 𝔞 𝔤 𝔬 𝔣 𝔟 ℭ X' x' by (rule assms(2))
  obtain f where f: f : X ↦_{iso ℭ} X'
    and j ∈_∘ ←•→_C(⦇Obj⦈) ⟹ x'(⦇NTMap⦈)(⦇j⦈) = f ∘_{A ℭ} x(⦇NTMap⦈)(⦇j⦈) for j
    by
      (
        elim cat-colim-ex-is-iso-arr'[
          OF x.is-cat-colimit-axioms x'.is-cat-colimit-axioms
          ]
      )
  then have x'(⦇NTMap⦈)(⦇𝔞_{SS}⦈) = f ∘_{A ℭ} x(⦇NTMap⦈)(⦇𝔞_{SS}⦈)
    and x'(⦇NTMap⦈)(⦇𝔟_{SS}⦈) = f ∘_{A ℭ} x(⦇NTMap⦈)(⦇𝔟_{SS}⦈)
    by (auto simp: cat-ss-cs-intros)
  with f show ?thesis using that by simp
qed
```

7 Equalizers and coequalizers as limits and colimits

7.1 Equalizer and coequalizer

7.1.1 Definition and elementary properties

See [2][6].

locale *is-cat-equalizer* =
  is-cat-limit α $\langle\Uparrow_C$ $(\mathfrak{a}_{PL}$ $F)$ $(\mathfrak{b}_{PL}$ $F)$ $F\rangle$ $\mathfrak{C}$ $\langle\Uparrow\!\!\to\!\!\Uparrow_{CF}$ $\mathfrak{C}$ $(\mathfrak{a}_{PL}$ $F)$ $(\mathfrak{b}_{PL}$ $F)$ F $\mathfrak{a}$ $\mathfrak{b}$ $F'\rangle$ E ε +
  F': *vsv* F'
  for α $\mathfrak{a}$ $\mathfrak{b}$ F F' $\mathfrak{C}$ E ε +
  assumes *cat-eq-F-in-Vset*[*cat-lim-cs-intros*]: $F \in_\circ Vset\ \alpha$
    and *cat-eq-F-ne*[*cat-lim-cs-intros*]: $F \neq 0$
    and *cat-eq-F'-vdomain*[*cat-lim-cs-simps*]: $\mathcal{D}_\circ\ F' = F$
    and *cat-eq-F'-app-is-arr*[*cat-lim-cs-intros*]: $\mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$

syntax *-is-cat-equalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} <_{CF.eq} '(\text{-},\text{-},\text{-},\text{-}') :/ \Uparrow_C \mapsto\!\!\mapsto_{C1} \text{-})\rangle$ [51, 51, 51, 51, 51, 51] 51)
syntax-consts *-is-cat-equalizer* $\rightleftharpoons$ *is-cat-equalizer*
translations $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},F,F') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-cat-equalizer α $\mathfrak{a}$ $\mathfrak{b}$ F F' $\mathfrak{C}$ E ε

locale *is-cat-coequalizer* =
  is-cat-colimit α $\langle\Uparrow_C$ $(\mathfrak{b}_{PL}$ $F)$ $(\mathfrak{a}_{PL}$ $F)$ $F\rangle$ $\mathfrak{C}$ $\langle\Uparrow\!\!\to\!\!\Uparrow_{CF}$ $\mathfrak{C}$ $(\mathfrak{b}_{PL}$ $F)$ $(\mathfrak{a}_{PL}$ $F)$ F $\mathfrak{b}$ $\mathfrak{a}$ $F'\rangle$ E ε +
  F': *vsv* F'
  for α $\mathfrak{a}$ $\mathfrak{b}$ F F' $\mathfrak{C}$ E ε +
  assumes *cat-coeq-F-in-Vset*[*cat-lim-cs-intros*]: $F \in_\circ Vset\ \alpha$
    and *cat-coeq-F-ne*[*cat-lim-cs-intros*]: $F \neq 0$
    and *cat-coeq-F'-vdomain*[*cat-lim-cs-simps*]: $\mathcal{D}_\circ\ F' = F$
    and *cat-coeq-F'-app-is-arr*[*cat-lim-cs-intros*]: $\mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$

syntax *-is-cat-coequalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ '(\text{-},\text{-},\text{-},\text{-}') >_{CF.coeq} \text{-} :/ \Uparrow_C \mapsto\!\!\mapsto_{C1} \text{-})\rangle$ [51, 51, 51, 51, 51, 51] 51)
syntax-consts *-is-cat-coequalizer* $\rightleftharpoons$ *is-cat-coequalizer*
translations $\varepsilon : (\mathfrak{a},\mathfrak{b},F,F') >_{CF.coeq} E : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-cat-coequalizer α $\mathfrak{a}$ $\mathfrak{b}$ F F' $\mathfrak{C}$ E ε

Rules.

lemma (**in** *is-cat-equalizer*) *is-cat-equalizer-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$
    and $E' = E$
    and $\mathfrak{a}' = \mathfrak{a}$
    and $\mathfrak{b}' = \mathfrak{b}$
    and $F'' = F$
    and $F''' = F'$
    and $\mathfrak{C}' = \mathfrak{C}$
  shows $\varepsilon : E' <_{CF.eq} (\mathfrak{a}',\mathfrak{b}',F'',F''') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha'} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-equalizer-axioms*)

mk-ide rf *is-cat-equalizer-def*[*unfolded is-cat-equalizer-axioms-def*]
  |*intro is-cat-equalizerI*|
  |*dest is-cat-equalizerD*[*dest*]|
  |*elim is-cat-equalizerE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-equalizerD*(*1*)

[6] https://en.wikipedia.org/wiki/Equaliser_(mathematics)

lemma (**in** *is-cat-coequalizer*) *is-cat-coequalizer-axioms'*[*cat-lim-cs-intros*]:
  assumes $\alpha' = \alpha$
    and $E' = E$
    and $\mathfrak{a}' = \mathfrak{a}$
    and $\mathfrak{b}' = \mathfrak{b}$
    and $F'' = F$
    and $F''' = F'$
    and $\mathfrak{C}' = \mathfrak{C}$
  shows $\varepsilon : (\mathfrak{a}', \mathfrak{b}', F'', F''') >_{CF.coeq} E' : \Uparrow_C \mapsto\mapsto_{C_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-coequalizer-axioms*)

mk-ide rf *is-cat-coequalizer-def*[*unfolded is-cat-coequalizer-axioms-def*]
  |*intro is-cat-coequalizerI*|
  |*dest is-cat-coequalizerD*[*dest*]|
  |*elim is-cat-coequalizerE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-coequalizerD*(*1*)

Elementary properties.

lemma (**in** *is-cat-equalizer*)
  cat-eq-$\mathfrak{a}$[*cat-lim-cs-intros*]: $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$
  and *cat-eq-*$\mathfrak{b}$[*cat-lim-cs-intros*]: $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$
proof–
  from *cat-eq-F-ne* **obtain** $\mathfrak{f}$ **where** $\mathfrak{f}$: $\mathfrak{f} \in_\circ F$ **by** *force*
  have $F'(\!|\mathfrak{f}|\!) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$ **by** (*rule cat-eq-F'-app-is-arr*[*OF* $\mathfrak{f}$])
  then show $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$ $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
qed

lemma (**in** *is-cat-coequalizer*)
  cat-coeq-$\mathfrak{a}$[*cat-lim-cs-intros*]: $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$
  and *cat-coeq-*$\mathfrak{b}$[*cat-lim-cs-intros*]: $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$
proof–
  from *cat-coeq-F-ne* **obtain** $\mathfrak{f}$ **where** $\mathfrak{f}$: $\mathfrak{f} \in_\circ F$ **by** *force*
  have $F'(\!|\mathfrak{f}|\!) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ **by** (*rule cat-coeq-F'-app-is-arr*[*OF* $\mathfrak{f}$])
  then show $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$ $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
qed

sublocale *is-cat-equalizer* $\subseteq$ *cf-parallel* α $\langle \mathfrak{a}_{PL}\ F \rangle$ $\langle \mathfrak{b}_{PL}\ F \rangle$ F $\mathfrak{a}$ $\mathfrak{b}$ F' $\mathfrak{C}$
  by (*intro cf-parallelI cat-parallelI*)
    (
      auto simp:
        cat-lim-cs-simps cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros
)

sublocale *is-cat-coequalizer* $\subseteq$ *cf-parallel* α $\langle \mathfrak{b}_{PL}\ F \rangle$ $\langle \mathfrak{a}_{PL}\ F \rangle$ F $\mathfrak{b}$ $\mathfrak{a}$ F' $\mathfrak{C}$
  by (*intro cf-parallelI cat-parallelI*)
    (
      auto simp:
        cat-lim-cs-simps cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros
)

Duality.

lemma (**in** *is-cat-equalizer*) *is-cat-coequalizer-op*:
  op-ntcf $\varepsilon : (\mathfrak{a}, \mathfrak{b}, F, F') >_{CF.coeq} E : \Uparrow_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
  by (*intro is-cat-coequalizerI*)
    (
      cs-concl
        cs-simp: *cat-lim-cs-simps cat-op-simps*

cs-intro: *V-cs-intros cat-op-intros cat-lim-cs-intros*
)+

lemma (**in** *is-cat-equalizer*) *is-cat-coequalizer-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows *op-ntcf* $\varepsilon : (\mathfrak{a},\mathfrak{b},F,F') >_{CF.coeq} E : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-coequalizer-op*)

lemmas [*cat-op-intros*] = *is-cat-equalizer.is-cat-coequalizer-op'*

lemma (**in** *is-cat-coequalizer*) *is-cat-equalizer-op*:
op-ntcf $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},F,F') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} op\text{-}cat\ \mathfrak{C}$
by (*intro is-cat-equalizerI*)
(
cs-concl
cs-simp: *cat-lim-cs-simps cat-op-simps*
cs-intro: *V-cs-intros cat-op-intros cat-lim-cs-intros*
)+

lemma (**in** *is-cat-coequalizer*) *is-cat-equalizer-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows *op-ntcf* $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},F,F') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-equalizer-op*)

lemmas [*cat-op-intros*] = *is-cat-coequalizer.is-cat-equalizer-op'*

Further properties.

lemma (**in** *category*) *cat-cf-parallel-*$\mathfrak{ab}$:
assumes *vsv* F'
and $F \in_\circ Vset\ \alpha$
and $\mathcal{D}_\circ\ F' = F$
and $\bigwedge \mathfrak{f}.\ \mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
and $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *cf-parallel* α $(\mathfrak{a}_{PL}\ F)$ $(\mathfrak{b}_{PL}\ F)$ F $\mathfrak{a}$ $\mathfrak{b}$ F' $\mathfrak{C}$
proof–
have $\mathfrak{a}_{PL}\ F \in_\circ Vset\ \alpha$ $\mathfrak{b}_{PL}\ F \in_\circ Vset\ \alpha$
by (*simp-all add*: *Axiom-of-Pairing* $\mathfrak{b}_{PL}$*-def* $\mathfrak{a}_{PL}$*-def assms*(*2*))
then show *?thesis*
by (*intro cf-parallelI cat-parallelI*)
(*simp-all add*: *assms cat-parallel-cs-intros cat-cs-intros*)
qed

lemma (**in** *category*) *cat-cf-parallel-*$\mathfrak{ba}$:
assumes *vsv* F'
and $F \in_\circ Vset\ \alpha$
and $\mathcal{D}_\circ\ F' = F$
and $\bigwedge \mathfrak{f}.\ \mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
and $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *cf-parallel* α $(\mathfrak{b}_{PL}\ F)$ $(\mathfrak{a}_{PL}\ F)$ F $\mathfrak{b}$ $\mathfrak{a}$ F' $\mathfrak{C}$
proof–
have $\mathfrak{a}_{PL}\ F \in_\circ Vset\ \alpha$ $\mathfrak{b}_{PL}\ F \in_\circ Vset\ \alpha$
by (*simp-all add*: *Axiom-of-Pairing* $\mathfrak{b}_{PL}$*-def* $\mathfrak{a}_{PL}$*-def assms*(*2*))
then show *?thesis*
by (*intro cf-parallelI cat-parallelI*)
(*simp-all add*: *assms cat-parallel-cs-intros cat-cs-intros*)
qed

lemma *cat-cone-cf-par-eps-NTMap-app*:
  assumes ε :
    $E <_{CF.cone} \Uparrow\!\!\rightarrow\!\Uparrow_{CF} \mathfrak{C}\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F\ \mathfrak{a}\ \mathfrak{b}\ F' :$
    $\Uparrow_C\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
    and *vsv* F'
    and $F \in_\circ Vset\ \alpha$
    and $\mathcal{D}_\circ\ F' = F$
    and $\bigwedge \mathfrak{f}.\ \mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
    and $\mathfrak{f} \in_\circ F$
  shows $\varepsilon(\!|NTMap|\!)(\!|\mathfrak{b}_{PL}\ F|\!) = F'(\!|\mathfrak{f}|\!) \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!)$
proof–
  let *?II* = $\langle \Uparrow_C\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F \rangle$
    and *?II-II* = $\langle \Uparrow\!\!\rightarrow\!\Uparrow_{CF} \mathfrak{C}\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F\ \mathfrak{a}\ \mathfrak{b}\ F' \rangle$
  interpret ε: *is-cat-cone* $\alpha\ E$ *?II* $\mathfrak{C}$ *?II-II* ε **by** (*rule assms*(*1*))
  from *assms*(*5*,*6*) **have** $\mathfrak{a}$: $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $\mathfrak{b}$: $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
  interpret *par*: *cf-parallel* α $\langle \mathfrak{a}_{PL}\ F \rangle$ $\langle \mathfrak{b}_{PL}\ F \rangle$ $F\ \mathfrak{a}\ \mathfrak{b}\ F'\ \mathfrak{C}$
    by (*intro* ε*.NTDom.HomCod.cat-cf-parallel*-$\mathfrak{ab}$ *assms* $\mathfrak{a}\ \mathfrak{b}$)
  from *assms*(*6*) **have** $\mathfrak{f}$: $\mathfrak{f} : \mathfrak{a}_{PL}\ F \mapsto_{\Uparrow_C\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F}\ \mathfrak{b}_{PL}\ F$
    by (*simp-all add*: *par.the-cat-parallel-is-arr*-$\mathfrak{ab}F$)
  note ε*.cat-cone-Comp-commute*[*cat-cs-simps del*]
  from ε*.ntcf-Comp-commute*[*OF* $\mathfrak{f}$] *assms*(*6*) **show** *?thesis*
    by
      (
        cs-prems
          cs-simp: *cat-parallel-cs-simps cat-cs-simps*
          cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)
qed

lemma *cat-cocone-cf-par-eps-NTMap-app*:
  assumes ε :
    $\Uparrow\!\!\rightarrow\!\Uparrow_{CF} \mathfrak{C}\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F\ \mathfrak{b}\ \mathfrak{a}\ F' >_{CF.cocone} E :$
    $\Uparrow_C\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
    and *vsv* F'
    and $F \in_\circ Vset\ \alpha$
    and $\mathcal{D}_\circ\ F' = F$
    and $\bigwedge \mathfrak{f}.\ \mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
    and $\mathfrak{f} \in_\circ F$
  shows $\varepsilon(\!|NTMap|\!)(\!|\mathfrak{b}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} F'(\!|\mathfrak{f}|\!)$
proof–
  let *?II* = $\langle \Uparrow_C\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F \rangle$
    and *?II-II* = $\langle \Uparrow\!\!\rightarrow\!\Uparrow_{CF} \mathfrak{C}\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F\ \mathfrak{b}\ \mathfrak{a}\ F' \rangle$
  interpret ε: *is-cat-cocone* $\alpha\ E$ *?II* $\mathfrak{C}$ *?II-II* ε **by** (*rule assms*(*1*))
  from *assms*(*5*,*6*)
  have $\mathfrak{a}$: $\mathfrak{a} \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $\mathfrak{b}$: $\mathfrak{b} \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $F'\mathfrak{f}$: $F'(\!|\mathfrak{f}|\!) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
    by *auto*
  interpret *par*: *cf-parallel* α $\langle \mathfrak{b}_{PL}\ F \rangle$ $\langle \mathfrak{a}_{PL}\ F \rangle$ $F\ \mathfrak{b}\ \mathfrak{a}\ F'\ \mathfrak{C}$
    by (*intro* ε*.NTDom.HomCod.cat-cf-parallel*-$\mathfrak{ba}$ *assms* $\mathfrak{a}\ \mathfrak{b}$)
  note ε*-NTMap-app* =
    cat-cone-cf-par-eps-NTMap-app[
      OF ε*.is-cat-cone-op*[*unfolded cat-op-simps*],
      unfolded cat-op-simps,
      OF assms(*2–6*),
      simplified
      ]
  from ε*-NTMap-app* $F'\mathfrak{f}$ **show** *?thesis*
    by

```
    (
      cs-concl cs-shallow
        cs-simp: cat-parallel-cs-simps category.op-cat-Comp[symmetric]
        cs-intro: cat-cs-intros cat-parallel-cs-intros
    )
qed

lemma (in is-cat-equalizer) cat-eq-eps-NTMap-app:
  assumes 𝔣 ∈ₒ F
  shows ε(|NTMap|)(|𝔟_PL F|) = F′(|𝔣|) ∘_A𝔈 ε(|NTMap|)(|𝔞_PL F|)
  by
    (
      intro cat-cone-cf-par-eps-NTMap-app[
        OF
          is-cat-cone-axioms
          F′.vsv-axioms
          cat-eq-F-in-Vset
          cat-eq-F′-vdomain
          cat-eq-F′-app-is-arr
          assms
        ]
    )+

lemma (in is-cat-coequalizer) cat-coeq-eps-NTMap-app:
  assumes 𝔣 ∈ₒ F
  shows ε(|NTMap|)(|𝔟_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A𝔈 F′(|𝔣|)
  by
    (
      intro cat-cocone-cf-par-eps-NTMap-app[
        OF is-cat-cocone-axioms
          F′.vsv-axioms
          cat-coeq-F-in-Vset
          cat-coeq-F′-vdomain
          cat-coeq-F′-app-is-arr
          assms
        ]
    )+

lemma (in is-cat-equalizer) cat-eq-Comp-eq:
  assumes 𝔤 ∈ₒ F and 𝔣 ∈ₒ F
  shows F′(|𝔤|) ∘_A𝔈 ε(|NTMap|)(|𝔞_PL F|) = F′(|𝔣|) ∘_A𝔈 ε(|NTMap|)(|𝔞_PL F|)
  using cat-eq-eps-NTMap-app[OF assms(1)] cat-eq-eps-NTMap-app[OF assms(2)]
  by auto

lemma (in is-cat-coequalizer) cat-coeq-Comp-eq:
  assumes 𝔤 ∈ₒ F and 𝔣 ∈ₒ F
  shows ε(|NTMap|)(|𝔞_PL F|) ∘_A𝔈 F′(|𝔤|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A𝔈 F′(|𝔣|)
  using cat-coeq-eps-NTMap-app[OF assms(1)] cat-coeq-eps-NTMap-app[OF assms(2)]
  by auto
```

7.1.2 Universal property

```
lemma is-cat-equalizerI′:
  assumes ε :
    E <_CF.cone ⇑→⇑_CF 𝔈 (𝔞_PL F) (𝔟_PL F) F 𝔞 𝔟 F′ :
    ⇑_C (𝔞_PL F) (𝔟_PL F) F ↦↦_Cα 𝔈
    and vsv F′
    and F ∈ₒ Vset α
```

```
    and 𝒟∘ F′ = F
    and ⋀𝔣. 𝔣 ∈∘ F ⟹ F′(|𝔣|) : 𝔞 ↦𝔈 𝔟
    and 𝔣 ∈∘ F
    and ⋀ε′ E′. ε′ :
      E′ <CF.cone ⇑→⇑CF 𝔈 (𝔞PL F) (𝔟PL F) F 𝔞 𝔟 F′ :
      ⇑C (𝔞PL F) (𝔟PL F) F ↦↦Cα 𝔈 ⟹
      ∃!f′. f′ : E′ ↦𝔈 E ∧ ε′(|NTMap|)(|𝔞PL F|) = ε(|NTMap|)(|𝔞PL F|) ∘A𝔈 f′
  shows ε : E <CF.eq (𝔞,𝔟,F,F′) : ⇑C ↦↦Cα 𝔈
proof–

  let ?II = ‹⇑C (𝔞PL F) (𝔟PL F) F›
    and ?II-II = ‹⇑→⇑CF 𝔈 (𝔞PL F) (𝔟PL F) F 𝔞 𝔟 F′›
  interpret ε: is-cat-cone α E ?II 𝔈 ?II-II ε by (rule assms(1))
  from assms(5,6) have 𝔞: 𝔞 ∈∘ 𝔈(|Obj|) and 𝔟: 𝔟 ∈∘ 𝔈(|Obj|) by auto
  interpret par: cf-parallel α ‹𝔞PL F› ‹𝔟PL F› F 𝔞 𝔟 F′ 𝔈
    by (intro ε.NTDom.HomCod.cat-cf-parallel-𝔞𝔟 assms 𝔞 𝔟) simp

  show ?thesis
  proof(intro is-cat-equalizerI is-cat-limitI assms(1–3))
    fix u′ r′ assume prems: u′ : r′ <CF.cone ?II-II : ?II ↦↦Cα 𝔈
    interpret u′: is-cat-cone α r′ ?II 𝔈 ?II-II u′ by (rule prems)
    from assms(7)[OF prems] obtain f′
      where f′: f′ : r′ ↦𝔈 E
        and u′-NTMap-app-𝔞: u′(|NTMap|)(|𝔞PL F|) = ε(|NTMap|)(|𝔞PL F|) ∘A𝔈 f′
        and unique-f′:
          ⋀f″.
            ⟦
              f″ : r′ ↦𝔈 E;
              u′(|NTMap|)(|𝔞PL F|) = ε(|NTMap|)(|𝔞PL F|) ∘A𝔈 f″
            ⟧ ⟹ f″ = f′
      by metis
    show ∃!f′. f′ : r′ ↦𝔈 E ∧ u′ = ε ⋆NTCF ntcf-const ?II 𝔈 f′
    proof(intro ex1I conjI; (elim conjE)?)
      show u′ = ε ⋆NTCF ntcf-const ?II 𝔈 f′
      proof(rule ntcf-eqI)
        show u′ : cf-const ?II 𝔈 r′ ↦CF ?II-II : ?II ↦↦Cα 𝔈
          by (rule u′.is-ntcf-axioms)
        from f′ show ε ⋆NTCF ntcf-const ?II 𝔈 f′ :
          cf-const ?II 𝔈 r′ ↦CF ?II-II : ?II ↦↦Cα 𝔈
          by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros )
        have dom-lhs: 𝒟∘ (u′(|NTMap|)) = ?II(|Obj|)
          unfolding cat-cs-simps by simp
        from f′ have dom-rhs:
          𝒟∘ ((ε ⋆NTCF ntcf-const ?II 𝔈 f′)(|NTMap|)) = ?II(|Obj|)
          by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
        show u′(|NTMap|) = (ε ⋆NTCF ntcf-const ?II 𝔈 f′)(|NTMap|)
        proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
          fix a assume prems′: a ∈∘ ?II(|Obj|)
          note [cat-parallel-cs-simps] =
            cat-cone-cf-par-eps-NTMap-app[
              OF u′.is-cat-cone-axioms assms(2–5), simplified
              ]
            cat-cone-cf-par-eps-NTMap-app[OF assms(1–5), simplified]
            u′-NTMap-app-𝔞
          from prems′ f′ assms(6) show
            u′(|NTMap|)(|a|) = (ε ⋆NTCF ntcf-const ?II 𝔈 f′)(|NTMap|)(|a|)
            by (elim the-cat-parallel-ObjE; simp only:)
              (
```

```
            cs-concl
              cs-simp: cat-parallel-cs-simps cat-cs-simps
              cs-intro: cat-cs-intros cat-parallel-cs-intros
          )
    qed (cs-concl cs-intro: V-cs-intros cat-cs-intros)+
  qed simp-all
  fix f'' assume prems'':
    f'': r' ↦_𝔈 E u' = ε ∙_NTCF ntcf-const ?II 𝔈 f''
  from prems''(2) have u'-NTMap-a:
    u'(|NTMap|)(|a|) = (ε ∙_NTCF ntcf-const ?II 𝔈 f'')(|NTMap|)(|a|)
    for a
    by simp
  have u'(|NTMap|)(|𝔞_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A𝔈 f''
    using u'-NTMap-a[of ‹𝔞_PL F›] prems''(1)
    by
      (
        cs-prems
          cs-simp: cat-parallel-cs-simps cat-cs-simps
          cs-intro: cat-parallel-cs-intros cat-cs-intros
      )
  from unique-f'[OF prems''(1) this] show f'' = f'.
 qed (rule f')
qed (use assms in fastforce)+

qed

lemma is-cat-coequalizerI':
  assumes ε :
    ⇑→⇑_CF 𝔈 (𝔟_PL F) (𝔞_PL F) F 𝔟 𝔞 F' >_CF.cocone E :
    ⇑_C (𝔟_PL F) (𝔞_PL F) F ↦↦_Cα 𝔈
    and vsv F'
    and F ∈_∘ Vset α
    and 𝒟_∘ F' = F
    and ⋀𝔣. 𝔣 ∈_∘ F ⟹ F'(|𝔣|) : 𝔟 ↦_𝔈 𝔞
    and 𝔣 ∈_∘ F
    and ⋀ε' E'. ε' :
      ⇑→⇑_CF 𝔈 (𝔟_PL F) (𝔞_PL F) F 𝔟 𝔞 F' >_CF.cocone E' :
      ⇑_C (𝔟_PL F) (𝔞_PL F) F ↦↦_Cα 𝔈 ⟹
      ∃!f'. f' : E ↦_𝔈 E' ∧ ε'(|NTMap|)(|𝔞_PL F|) = f' ∘_A𝔈 ε(|NTMap|)(|𝔞_PL F|)
  shows ε : (𝔞,𝔟,F,F') >_CF.coeq E : ⇑_C ↦↦_Cα 𝔈
proof–

  let ?op-II = ‹⇑_C (𝔟_PL F) (𝔞_PL F) F›
    and ?op-II-II = ‹⇑→⇑_CF 𝔈 (𝔟_PL F) (𝔞_PL F) F 𝔟 𝔞 F'›
    and ?II = ‹⇑_C (𝔞_PL F) (𝔟_PL F) F›
    and ?II-II = ‹⇑→⇑_CF (op-cat 𝔈) (𝔞_PL F) (𝔟_PL F) F 𝔞 𝔟 F'›
  interpret ε: is-cat-cocone α E ?op-II 𝔈 ?op-II-II ε by (rule assms(1))
  from assms(5,6) have 𝔞: 𝔞 ∈_∘ 𝔈(|Obj|) and 𝔟: 𝔟 ∈_∘ 𝔈(|Obj|) by auto
  interpret par: cf-parallel α ‹𝔟_PL F› ‹𝔞_PL F› F 𝔟 𝔞 F' 𝔈
    by (intro ε.NTDom.HomCod.cat-cf-parallel-𝔟𝔞 assms 𝔞 𝔟) simp

  interpret op-par: cf-parallel α ‹𝔞_PL F› ‹𝔟_PL F› F 𝔞 𝔟 F' ‹op-cat 𝔈›
    by (rule par.cf-parallel-op)
  have assms-4':
    ∃!f'. f' : E ↦_𝔈 E' ∧ ε'(|NTMap|)(|𝔞_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A op-cat 𝔈 f'
    if ε' : E' <_CF.cone ?II-II : ?II ↦↦_Cα op-cat 𝔈 for ε' E'
  proof–
    have [cat-op-simps]:
```

```
    f′ : E ↦_𝔈 E′ ∧ ε′(|NTMap|)(|𝔞_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A op-cat 𝔈 f′ ⟷
      f′ : E ↦_𝔈 E′ ∧ ε′(|NTMap|)(|𝔞_PL F|) = f′ ∘_A𝔈 ε(|NTMap|)(|𝔞_PL F|)
    for f′
    by (intro iffI conjI; (elim conjE)?)
      (
        cs-concl cs-shallow
          cs-simp: category.op-cat-Comp[symmetric] cat-op-simps cat-cs-simps
          cs-intro: cat-cs-intros cat-parallel-cs-intros
      )+
  interpret ε′: is-cat-cone α E′ ?II ‹op-cat 𝔈› ?II-II ε′ by (rule that)
  show ?thesis
    unfolding cat-op-simps
    by
      (
        rule assms(7)[
          OF ε′.is-cat-cocone-op[unfolded cat-op-simps],
          unfolded cat-op-simps
          ]
      )
qed
interpret op-ε: is-cat-equalizer α 𝔞 𝔟 F F′ ‹op-cat 𝔈› E ‹op-ntcf ε›
  by
    (
      rule
        is-cat-equalizerI′
          [
            OF ε.is-cat-cone-op[unfolded cat-op-simps],
            unfolded cat-op-simps,
            OF assms(2–6) assms-4′,
            simplified
          ]
    )
show ?thesis by (rule op-ε.is-cat-coequalizer-op[unfolded cat-op-simps])

qed

lemma (in is-cat-equalizer) cat-eq-unique-cone:
  assumes ε′ :
    E′ <_CF.cone ⇑→⇑_CF 𝔈 (𝔞_PL F) (𝔟_PL F) F 𝔞 𝔟 F′ : ⇑_C (𝔞_PL F) (𝔟_PL F) F ↦↦_Cα 𝔈
    (is ‹ε′ : E′ <_CF.cone ?II-II : ?II ↦↦_Cα 𝔈›)
  shows ∃!f′. f′ : E′ ↦_𝔈 E ∧ ε′(|NTMap|)(|𝔞_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A𝔈 f′
proof–

  interpret ε′: is-cat-cone α E′ ?II 𝔈 ?II-II ε′ by (rule assms(1))
  from cat-lim-ua-fo[OF assms(1)] obtain f′ where f′: f′ : E′ ↦_𝔈 E
    and ε′-def: ε′ = ε ∙_NTCF ntcf-const ?II 𝔈 f′
    and unique:
      [[ f″ : E′ ↦_𝔈 E; ε′ = ε ∙_NTCF ntcf-const ?II 𝔈 f″ ]] ⟹ f″ = f′
    for f″
    by auto
  from cat-eq-F-ne obtain 𝔣 where 𝔣: 𝔣 ∈_∘ F by force

  show ?thesis
  proof(intro ex1I conjI; (elim conjE)?)
    show f′: f′ : E′ ↦_𝔈 E by (rule f′)
    from ε′-def have ε′(|NTMap|)(|𝔞_PL F|) = (ε ∙_NTCF ntcf-const ?II 𝔈 f′)(|NTMap|)(|𝔞_PL F|)
      by simp
    from this f′ show ε′-NTMap-app-I: ε′(|NTMap|)(|𝔞_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_A𝔈 f′
```

```
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-parallel-cs-intros
        )
    fix f'' assume prems:
      f'' : E' ↦_ℭ E ε'(|NTMap|)(|𝔞_PL F|) = ε(|NTMap|)(|𝔞_PL F|) ∘_Aℭ f''
    have ε' = ε ⋆_NTCF ntcf-const ?II ℭ f''
    proof(rule ntcf-eqI[OF ])
      show ε' : cf-const ?II ℭ E' ↦_CF ?II-II : ?II ↦↦_Cα ℭ
        by (rule ε'.is-ntcf-axioms)
      from f' prems(1) show ε ⋆_NTCF ntcf-const ?II ℭ f'' :
        cf-const ?II ℭ E' ↦_CF ?II-II : ?II ↦↦_Cα ℭ
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      show ε'(|NTMap|) = (ε ⋆_NTCF ntcf-const ?II ℭ f'')(|NTMap|)
      proof(rule vsv-eqI, unfold cat-cs-simps)
        show vsv ((ε ⋆_NTCF ntcf-const ?II ℭ f'')(|NTMap|))
          by (cs-concl cs-intro: cat-cs-intros)
        from prems(1) show ?II(|Obj|) = 𝒟_∘ ((ε ⋆_NTCF ntcf-const ?II ℭ f'')(|NTMap|))
          by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
        fix a assume prems': a ∈_∘ ?II(|Obj|)
        note [cat-cs-simps] =
          cat-eq-eps-NTMap-app[OF ϱ]
          cat-cone-cf-par-eps-NTMap-app
            [
              OF
                ε'.is-cat-cone-axioms
                F'.vsv-axioms
                cat-eq-F-in-Vset
                cat-eq-F'-vdomain
                cat-eq-F'-app-is-arr ϱ,
              simplified
            ]
        from prems' prems(1) ϱ have [cat-cs-simps]:
          ε'(|NTMap|)(|a|) = ε(|NTMap|)(|a|) ∘_Aℭ f''
          by (elim the-cat-parallel-ObjE; simp only:)
            (
                cs-concl
                  cs-simp: cat-cs-simps cat-parallel-cs-simps prems(2)
                  cs-intro: cat-cs-intros cat-parallel-cs-intros
            )+
        from prems' prems show
          ε'(|NTMap|)(|a|) = (ε ⋆_NTCF ntcf-const ?II ℭ f'')(|NTMap|)(|a|)
          by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      qed auto
    qed simp-all
    from unique[OF prems(1) this] show f'' = f' .
  qed

qed

lemma (in is-cat-equalizer) cat-eq-unique:
  assumes ε' : E' <_CF.eq (𝔞,𝔟,F,F') : ⇑_C ↦↦_Cα ℭ
  shows
    ∃!f'. f' : E' ↦_ℭ E ∧ ε' = ε ⋆_NTCF ntcf-const (⇑_C (𝔞_PL F) (𝔟_PL F) F) ℭ f'
  by (rule cat-lim-unique[OF is-cat-equalizerD(1)[OF assms]])

lemma (in is-cat-equalizer) cat-eq-unique':
```

assumes $\varepsilon' : E' <_{CF.eq} (\mathfrak{a},\mathfrak{b},F,F') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} f'$
proof–
interpret ε': *is-cat-equalizer* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ F\ F'\ \mathfrak{C}\ E'\ \varepsilon'$ **by** (*rule assms*(*1*))
show *?thesis* **by** (*rule cat-eq-unique-cone*[*OF* ε'.*is-cat-cone-axioms*])
qed

lemma (**in** *is-cat-coequalizer*) *cat-coeq-unique-cocone*:
assumes ε' :
$\Uparrow\!\!\to\!\Uparrow_{CF}\ \mathfrak{C}\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F\ \mathfrak{b}\ \mathfrak{a}\ F' >_{CF.cocone} E'$:
$\Uparrow_C\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
(**is** $\langle\varepsilon'$: *?II-II* $>_{CF.cocone} E'$: *?II* $\mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}\rangle$)
shows $\exists! f'.\ f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = f' \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!)$
proof–
interpret ε': *is-cat-cocone* $\alpha\ E'$ *?II* $\mathfrak{C}$ *?II-II* ε' **by** (*rule assms*(*1*))
have [*cat-op-simps*]:
$f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\,op\text{-}cat\ \mathfrak{C}} f' \longleftrightarrow$
$f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = f' \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!)$
for f'
by (*intro iffI conjI*; (*elim conjE*)*?*)
(
cs-concl **cs-shallow**
cs-simp: *category.op-cat-Comp*[*symmetric*] *cat-op-simps cat-cs-simps*
cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)+
show *?thesis*
by
(
rule is-cat-equalizer.cat-eq-unique-cone[
OF is-cat-equalizer-op ε'.*is-cat-cone-op*[*unfolded cat-op-simps*],
unfolded cat-op-simps
]
)
qed

lemma (**in** *is-cat-coequalizer*) *cat-coeq-unique*:
assumes $\varepsilon' : (\mathfrak{a},\mathfrak{b},F,F') >_{CF.coeq} E' : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.$
$f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon' = ntcf\text{-}const\ (\Uparrow_C\ (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F)\ \mathfrak{C}\ f' \cdot_{NTCF} \varepsilon$
by (*rule cat-colim-unique*[*OF is-cat-coequalizerD*(*1*)[*OF assms*]])

lemma (**in** *is-cat-coequalizer*) *cat-coeq-unique'*:
assumes $\varepsilon' : (\mathfrak{a},\mathfrak{b},F,F') >_{CF.coeq} E' : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'.\ f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = f' \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!)$
proof–
interpret ε': *is-cat-coequalizer* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ F\ F'\ \mathfrak{C}\ E'\ \varepsilon'$ **by** (*rule assms*(*1*))
show *?thesis* **by** (*rule cat-coeq-unique-cocone*[*OF* ε'.*is-cat-cocone-axioms*])
qed

lemma *cat-equalizer-ex-is-iso-arr*:
assumes $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},F,F') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
and $\varepsilon' : E' <_{CF.eq} (\mathfrak{a},\mathfrak{b},F,F') : \Uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : E' \mapsto_{iso\mathfrak{C}} E$
and $\varepsilon' = \varepsilon \cdot_{NTCF} ntcf\text{-}const\ (\Uparrow_C\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F)\ \mathfrak{C}\ f$
proof–
interpret ε: *is-cat-equalizer* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ F\ F'\ \mathfrak{C}\ E\ \varepsilon$ **by** (*rule assms*(*1*))
interpret ε': *is-cat-equalizer* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ F\ F'\ \mathfrak{C}\ E'\ \varepsilon'$ **by** (*rule assms*(*2*))
from *that* **show** *?thesis*

```
    by
      (
        elim cat-lim-ex-is-iso-arr[
          OF ε.is-cat-limit-axioms ε′.is-cat-limit-axioms
          ]
      )
qed

lemma cat-equalizer-ex-is-iso-arr′:
  assumes ε : E <CF.eq (𝔞,𝔟,F,F′) : ⇑C ↦↦Cα ℭ
    and ε′ : E′ <CF.eq (𝔞,𝔟,F,F′) : ⇑C ↦↦Cα ℭ
  obtains f where f : E′ ↦iso ℭ E
    and ε′(⌊NTMap⌋)(⌊𝔞PL F⌋) = ε(⌊NTMap⌋)(⌊𝔞PL F⌋) ∘A ℭ f
    and ε′(⌊NTMap⌋)(⌊𝔟PL F⌋) = ε(⌊NTMap⌋)(⌊𝔟PL F⌋) ∘A ℭ f
proof–
  interpret ε: is-cat-equalizer α 𝔞 𝔟 F F′ ℭ E ε by (rule assms(1))
  interpret ε′: is-cat-equalizer α 𝔞 𝔟 F F′ ℭ E′ ε′ by (rule assms(2))
  obtain f where f: f : E′ ↦iso ℭ E
    and j ∈∘ ⇑C (𝔞PL F) (𝔟PL F) F(⌊Obj⌋) ⟹ ε′(⌊NTMap⌋)(⌊j⌋) = ε(⌊NTMap⌋)(⌊j⌋) ∘A ℭ f for j
    by
      (
        elim cat-lim-ex-is-iso-arr′[
          OF ε.is-cat-limit-axioms ε′.is-cat-limit-axioms
          ]
      )
  then have
    ε′(⌊NTMap⌋)(⌊𝔞PL F⌋) = ε(⌊NTMap⌋)(⌊𝔞PL F⌋) ∘A ℭ f
    ε′(⌊NTMap⌋)(⌊𝔟PL F⌋) = ε(⌊NTMap⌋)(⌊𝔟PL F⌋) ∘A ℭ f
    unfolding the-cat-parallel-components by auto
  with f show ?thesis using that by simp
qed

lemma cat-coequalizer-ex-is-iso-arr:
  assumes ε : (𝔞,𝔟,F,F′) >CF.coeq E : ⇑C ↦↦Cα ℭ
    and ε′ : (𝔞,𝔟,F,F′) >CF.coeq E′ : ⇑C ↦↦Cα ℭ
  obtains f where f : E ↦iso ℭ E′
    and ε′ = ntcf-const (⇑C (𝔟PL F) (𝔞PL F) F)  ℭ f ·NTCF ε
proof–
  interpret ε: is-cat-coequalizer α 𝔞 𝔟 F F′ ℭ E ε by (rule assms(1))
  interpret ε′: is-cat-coequalizer α 𝔞 𝔟 F F′ ℭ E′ ε′ by (rule assms(2))
  from that show ?thesis
    by
      (
        elim cat-colim-ex-is-iso-arr[
          OF ε.is-cat-colimit-axioms ε′.is-cat-colimit-axioms
          ]
      )
qed

lemma cat-coequalizer-ex-is-iso-arr′:
  assumes ε : (𝔞,𝔟,F,F′) >CF.coeq E : ⇑C ↦↦Cα ℭ
    and ε′ : (𝔞,𝔟,F,F′) >CF.coeq E′ : ⇑C ↦↦Cα ℭ
  obtains f where f : E ↦iso ℭ E′
    and ε′(⌊NTMap⌋)(⌊𝔞PL F⌋) = f ∘A ℭ ε(⌊NTMap⌋)(⌊𝔞PL F⌋)
    and ε′(⌊NTMap⌋)(⌊𝔟PL F⌋) = f ∘A ℭ ε(⌊NTMap⌋)(⌊𝔟PL F⌋)
proof–
  interpret ε: is-cat-coequalizer α 𝔞 𝔟 F F′ ℭ E ε by (rule assms(1))
  interpret ε′: is-cat-coequalizer α 𝔞 𝔟 F F′ ℭ E′ ε′ by (rule assms(2))
```

obtain f **where** f: $f : E \mapsto_{iso\mathfrak{C}} E'$
and $j \in_\circ \Uparrow_C (\mathfrak{b}_{PL}\ F)\ (\mathfrak{a}_{PL}\ F)\ F(\!|Obj|\!) \Longrightarrow \varepsilon'(\!|NTMap|\!)(\!|j|\!) = f \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|j|\!)$ **for** j
by
(
elim cat-colim-ex-is-iso-arr'[
OF ε.is-cat-colimit-axioms ε'.is-cat-colimit-axioms
]
)
then have
$\varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = f \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!)$
$\varepsilon'(\!|NTMap|\!)(\!|\mathfrak{b}_{PL}\ F|\!) = f \circ_{A\mathfrak{C}} \varepsilon(\!|NTMap|\!)(\!|\mathfrak{b}_{PL}\ F|\!)$
unfolding *the-cat-parallel-components* **by** *auto*
with f **show** *?thesis* **using** *that* **by** *simp*
qed

7.1.3 Further properties

lemma (in *is-cat-equalizer*) *cat-eq-is-monic-arr*:
— See subsection 3.3 in [3].
$\varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) : E \mapsto_{mon\mathfrak{C}} \mathfrak{a}$
proof(*intro is-monic-arrI*)
show $\varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) : E \mapsto_{\mathfrak{C}} \mathfrak{a}$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-parallel-cs-simps*
cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)
fix $f\ g\ a$
assume *prems*:
$f : a \mapsto_{\mathfrak{C}} E$
$g : a \mapsto_{\mathfrak{C}} E$
$\varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} f = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} g$
define ε' **where** $\varepsilon' = \varepsilon \cdot_{NTCF} ntcf\text{-}const\ (\Uparrow_C\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F)\ \mathfrak{C}\ f$
from *prems*(*1*) **have** ε' :
$a <_{CF.cone} \Uparrow\!\!\rightarrow\!\!\Uparrow_{CF} \mathfrak{C}\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F\ \mathfrak{a}\ \mathfrak{b}\ F'$:
$\Uparrow_C\ (\mathfrak{a}_{PL}\ F)\ (\mathfrak{b}_{PL}\ F)\ F \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
unfolding *ε'-def*
by (*cs-concl* **cs-shallow cs-intro**: *is-cat-coneI cat-cs-intros*)
from *cat-eq-unique-cone*[*OF this*] **obtain** f'
where f': $f' : a \mapsto_{\mathfrak{C}} E$
and ε'-$\mathfrak{a}$: $\varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} f'$
and *unique-f'*: $\bigwedge f''$.
$[\![\ f'' : a \mapsto_{\mathfrak{C}} E;\ \varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} f''\ ]\!] \Longrightarrow$
$f'' = f'$
by *meson*
from *prems*(*1*) **have** *unique-f*: $\varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} f$
unfolding *ε'-def*
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-parallel-cs-intros*
)
from *prems*(*1*) **have** *unique-g*: $\varepsilon'(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) = \varepsilon(\!|NTMap|\!)(\!|\mathfrak{a}_{PL}\ F|\!) \circ_{A\mathfrak{C}} g$
unfolding *ε'-def*
by
(
cs-concl

```
      cs-simp: prems(3) cat-cs-simps
      cs-intro: cat-cs-intros cat-parallel-cs-intros
    )
  show f = g
    by
      (
        rule unique-f'
          [
            OF prems(1) unique-f,
            unfolded unique-f'[OF prems(2) unique-g, symmetric]
          ]
      )
qed

lemma (in is-cat-coequalizer) cat-coeq-is-epic-arr:
  ε(⁅NTMap⁆)(⁅𝔞PL F⁆) : 𝔞 ↦epi𝔈 E
  by
    (
      rule is-cat-equalizer.cat-eq-is-monic-arr[
        OF is-cat-equalizer-op, unfolded cat-op-simps
        ]
    )
```

7.2 Equalizer and coequalizer for two arrows

7.2.1 Definition and elementary properties

See [2][7].

```
locale is-cat-equalizer-2 =
  is-cat-limit α ‹↑↑C 𝔞PL2 𝔟PL2 𝔤PL 𝔣PL› 𝔈 ‹↑↑→↑↑CF 𝔈 𝔞PL2 𝔟PL2 𝔤PL 𝔣PL 𝔞 𝔟 𝔤 𝔣› E ε
  for α 𝔞 𝔟 𝔤 𝔣 𝔈 E ε +
  assumes cat-eq-𝔤[cat-lim-cs-intros]: 𝔤 : 𝔞 ↦𝔈 𝔟
    and cat-eq-𝔣[cat-lim-cs-intros]: 𝔣 : 𝔞 ↦𝔈 𝔟

syntax -is-cat-equalizer-2 :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ bool
  (‹(- :/ - <CF.eq '(-,-,-,-') :/ ↑↑C ↦↦C1 -)› [51, 51, 51, 51, 51, 51] 51)
syntax-consts -is-cat-equalizer-2 ⇌ is-cat-equalizer-2
translations ε : E <CF.eq (𝔞,𝔟,𝔤,𝔣) : ↑↑C ↦↦Cα 𝔈 ⇌
  CONST is-cat-equalizer-2 α 𝔞 𝔟 𝔤 𝔣 𝔈 E ε

locale is-cat-coequalizer-2 =
  is-cat-colimit
    α ‹↑↑C 𝔟PL2 𝔞PL2 𝔣PL 𝔤PL› 𝔈 ‹↑↑→↑↑CF 𝔈 𝔟PL2 𝔞PL2 𝔣PL 𝔤PL 𝔟 𝔞 𝔣 𝔤› E ε
  for α 𝔞 𝔟 𝔤 𝔣 𝔈 E ε +
  assumes cat-coeq-𝔤[cat-lim-cs-intros]: 𝔤 : 𝔟 ↦𝔈 𝔞
    and cat-coeq-𝔣[cat-lim-cs-intros]: 𝔣 : 𝔟 ↦𝔈 𝔞

syntax -is-cat-coequalizer-2 :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ bool
  (‹(- :/ '(-,-,-,-') >CF.coeq - :/ ↑↑C ↦↦C1 -)› [51, 51, 51, 51, 51, 51] 51)
syntax-consts -is-cat-coequalizer-2 ⇌ is-cat-coequalizer-2
translations ε : (𝔞,𝔟,𝔤,𝔣) >CF.coeq E : ↑↑C ↦↦Cα 𝔈 ⇌
  CONST is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 𝔈 E ε
```

Rules.

```
lemma (in is-cat-equalizer-2) is-cat-equalizer-2-axioms'[cat-lim-cs-intros]:
  assumes α' = α
```

[7] https://en.wikipedia.org/wiki/Equaliser_(mathematics)

and $E' = E$
and $\mathfrak{a}' = \mathfrak{a}$
and $\mathfrak{b}' = \mathfrak{b}$
and $\mathfrak{g}' = \mathfrak{g}$
and $\mathfrak{f}' = \mathfrak{f}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\varepsilon : E' <_{CF.eq} (\mathfrak{a}',\mathfrak{b}',\mathfrak{g}',\mathfrak{f}') : \Uparrow_C \mapsto\mapsto_{C_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-equalizer-2-axioms*)

mk-ide rf *is-cat-equalizer-2-def*[*unfolded is-cat-equalizer-2-axioms-def*]
|*intro is-cat-equalizer-2I*|
|*dest is-cat-equalizer-2D*[*dest*]|
|*elim is-cat-equalizer-2E*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-equalizer-2D*(*1*)

lemma (**in** *is-cat-coequalizer-2*) *is-cat-coequalizer-2-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$
and $E' = E$
and $\mathfrak{a}' = \mathfrak{a}$
and $\mathfrak{b}' = \mathfrak{b}$
and $\mathfrak{g}' = \mathfrak{g}$
and $\mathfrak{f}' = \mathfrak{f}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\varepsilon : (\mathfrak{a}',\mathfrak{b}',\mathfrak{g}',\mathfrak{f}') >_{CF.coeq} E' : \Uparrow_C \mapsto\mapsto_{C_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-coequalizer-2-axioms*)

mk-ide rf *is-cat-coequalizer-2-def*[*unfolded is-cat-coequalizer-2-axioms-def*]
|*intro is-cat-coequalizer-2I*|
|*dest is-cat-coequalizer-2D*[*dest*]|
|*elim is-cat-coequalizer-2E*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-coequalizer-2D*(*1*)

Helper lemmas.

lemma *cat-eq-F'-helper*:
$(\lambda f \in_\circ set\ \{\mathfrak{f}_{PL},\ \mathfrak{g}_{PL}\}.\ (f = \mathfrak{g}_{PL}\ ?\ \mathfrak{g} : \mathfrak{f})) =$
$(\lambda f \in_\circ set\ \{\mathfrak{f}_{PL},\ \mathfrak{g}_{PL}\}.\ (f = \mathfrak{f}_{PL}\ ?\ \mathfrak{f} : \mathfrak{g}))$
using *cat-PL2-gf* **by** (*simp add*: *VLambda-vdoubleton*)

Elementary properties.

sublocale *is-cat-equalizer-2* $\subseteq$ *cf-parallel-2* α $\mathfrak{a}_{PL2}$ $\mathfrak{b}_{PL2}$ $\mathfrak{g}_{PL}$ $\mathfrak{f}_{PL}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{C}$
by (*intro cf-parallel-2I cat-parallel-2I*)
(*simp-all add*: *cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros*)

sublocale *is-cat-coequalizer-2* $\subseteq$ *cf-parallel-2* α $\mathfrak{b}_{PL2}$ $\mathfrak{a}_{PL2}$ $\mathfrak{f}_{PL}$ $\mathfrak{g}_{PL}$ $\mathfrak{b}$ $\mathfrak{a}$ $\mathfrak{f}$ $\mathfrak{g}$ $\mathfrak{C}$
by (*intro cf-parallel-2I cat-parallel-2I*)
(
auto simp:
cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros
cat-PL2-ineq[*symmetric*]
)

lemma (**in** *is-cat-equalizer-2*) *cat-equalizer-2-is-cat-equalizer*:
ε :
$E <_{CF.eq} (\mathfrak{a},\mathfrak{b},set\ \{\mathfrak{g}_{PL},\ \mathfrak{f}_{PL}\},(\lambda f \in_\circ set\ \{\mathfrak{g}_{PL},\ \mathfrak{f}_{PL}\}.\ (f = \mathfrak{f}_{PL}\ ?\ \mathfrak{f} : \mathfrak{g}))) :$
$\Uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by

```
    (
      intro is-cat-equalizerI,
      rule is-cat-limit-axioms[
        unfolded the-cf-parallel-2-def the-cat-parallel-2-def 𝔞_PL2-def 𝔟_PL2-def
        ]
    )
    (auto simp: Limit-vdoubleton-in-VsetI cat-parallel-cs-intros)

lemma (in is-cat-coequalizer-2) cat-coequalizer-2-is-cat-coequalizer:
  ε :
    (𝔞,𝔟,set {𝔤_PL, 𝔣_PL},(λf∈₀set {𝔤_PL, 𝔣_PL}. (f = 𝔣_PL ? 𝔣 : 𝔤))) >_CF.coeq E :
    ⇑_C ↦↦_Cα ℭ
proof
  (
    intro is-cat-coequalizerI,
    fold the-cf-parallel-2-def the-cat-parallel-2-def 𝔞_PL2-def 𝔟_PL2-def
  )
  show ε :
    ↑↑→↑↑_CF ℭ 𝔟_PL2 𝔞_PL2 𝔤_PL 𝔣_PL 𝔟 𝔞 𝔤 𝔣 >_CF.colim E :
    ↑↑_C 𝔟_PL2 𝔞_PL2 𝔤_PL 𝔣_PL ↦↦_Cα ℭ
    by
      (
        subst the-cat-parallel-2-commute,
        subst cf-parallel-2-the-cf-parallel-2-commute[symmetric]
      )
      (intro is-cat-colimit-axioms)
qed (auto simp: Limit-vdoubleton-in-VsetI cat-parallel-cs-intros)

lemma cat-equalizer-is-cat-equalizer-2:
  assumes ε :
    E <_CF.eq (𝔞,𝔟,set {𝔤_PL, 𝔣_PL},(λf∈₀set {𝔤_PL, 𝔣_PL}. (f = 𝔣_PL ? 𝔣 : 𝔤))) :
    ⇑_C ↦↦_Cα ℭ
  shows ε : E <_CF.eq (𝔞,𝔟,𝔤,𝔣) : ↑↑_C ↦↦_Cα ℭ
proof–
  interpret ε: is-cat-equalizer
    α 𝔞 𝔟 ‹set {𝔤_PL, 𝔣_PL}› ‹(λf∈₀set {𝔤_PL, 𝔣_PL}. (f = 𝔣_PL ? 𝔣 : 𝔤))› ℭ E ε
    by (rule assms)
  have 𝔣_PL: 𝔣_PL ∈₀ set {𝔤_PL, 𝔣_PL} and 𝔤_PL: 𝔤_PL ∈₀ set {𝔤_PL, 𝔣_PL} by auto
  show ?thesis
    using ε.cat-eq-F'-app-is-arr[OF 𝔤_PL] ε.cat-eq-F'-app-is-arr[OF 𝔣_PL]
    by
      (
        intro
          is-cat-equalizer-2I
          ε.is-cat-limit-axioms
            [
              folded
                the-cf-parallel-2-def the-cat-parallel-2-def 𝔞_PL2-def 𝔟_PL2-def
            ]
      )
      (auto simp: cat-PL2-𝔤𝔣)
qed

lemma cat-coequalizer-is-cat-coequalizer-2:
  assumes ε :
    (𝔞,𝔟,set {𝔤_PL, 𝔣_PL},(λf∈₀set {𝔤_PL, 𝔣_PL}. (f = 𝔣_PL ? 𝔣 : 𝔤))) >_CF.coeq E :
    ⇑_C ↦↦_Cα ℭ
  shows ε : (𝔞,𝔟,𝔤,𝔣) >_CF.coeq E : ↑↑_C ↦↦_Cα ℭ
```

proof–
interpret *is-cat-coequalizer*
α $\mathfrak{a}$ $\mathfrak{b}$ ‹*set* $\{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$› ‹$(\lambda f \in_{\circ} \textit{set}\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}.\ (f = \mathfrak{f}_{PL}\ ?\ \mathfrak{f} : \mathfrak{g}))$› $\mathfrak{C}$ E ε
by (*rule assms*)
interpret *cf-parallel-2* α $\mathfrak{b}_{PL2}$ $\mathfrak{a}_{PL2}$ $\mathfrak{g}_{PL}$ $\mathfrak{f}_{PL}$ $\mathfrak{b}$ $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{C}$
by
(
rule cf-parallel-is-cf-parallel-2[
OF cf-parallel-axioms cat-PL2-$\mathfrak{gf}$, *folded* $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*
]
)
show $\varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by
(
intro is-cat-coequalizer-2I,
subst the-cat-parallel-2-commute,
subst cf-parallel-2-the-cf-parallel-2-commute[*symmetric*],
rule is-cat-colimit-axioms[
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def the-cat-parallel-2-def the-cf-parallel-2-def*
]
)
(*simp-all add*: *cf-parallel-*$\mathfrak{f}'$ *cf-parallel-*$\mathfrak{g}'$)
qed

Duality.

lemma (**in** *is-cat-equalizer-2*) *is-cat-coequalizer-2-op*:
op-ntcf $\varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
unfolding *is-cat-equalizer-def*
by
(
rule cat-coequalizer-is-cat-coequalizer-2
[
OF is-cat-equalizer.is-cat-coequalizer-op[
OF cat-equalizer-2-is-cat-equalizer
]
]
)

lemma (**in** *is-cat-equalizer-2*) *is-cat-coequalizer-2-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-coequalizer-2-op*)

lemmas [*cat-op-intros*] = *is-cat-equalizer-2.is-cat-coequalizer-2-op'*

lemma (**in** *is-cat-coequalizer-2*) *is-cat-equalizer-2-op*:
op-ntcf $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
unfolding *is-cat-coequalizer-def*
by
(
rule cat-equalizer-is-cat-equalizer-2
[
OF is-cat-coequalizer.is-cat-equalizer-op[
OF cat-coequalizer-2-is-cat-coequalizer
]
]
)

lemma (**in** *is-cat-coequalizer-2*) *is-cat-equalizer-2-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows *op-ntcf* $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-equalizer-2-op*)

lemmas [*cat-op-intros*] = *is-cat-coequalizer-2.is-cat-equalizer-2-op'*

Further properties.

lemma (**in** *category*) *cat-cf-parallel-2-cat-equalizer*:
assumes $\mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$ **and** $\mathfrak{f} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
shows *cf-parallel-2* $\alpha\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{C}$
using *assms*
by (*intro cf-parallel-2I cat-parallel-2I*)
(*auto simp*: *cat-parallel-cs-intros cat-cs-intros*)

lemma (**in** *category*) *cat-cf-parallel-2-cat-coequalizer*:
assumes $\mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ **and** $\mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
shows *cf-parallel-2* $\alpha\ \mathfrak{b}_{PL2}\ \mathfrak{a}_{PL2}\ \mathfrak{f}_{PL}\ \mathfrak{g}_{PL}\ \mathfrak{b}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{g}\ \mathfrak{C}$
using *assms*
by (*intro cf-parallel-2I cat-parallel-2I*)
(*simp-all add*: *cat-parallel-cs-intros cat-cs-intros cat-PL2-ineq*[*symmetric*])

lemma *cat-cone-cf-par-2-eps-NTMap-app*:
assumes ε :
$E <_{CF.cone} \uparrow\uparrow\to\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f} : \uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
and $\mathfrak{f} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
shows
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
proof–
let *?II* = $\langle \uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \rangle$
and *?II-II* = $\langle \uparrow\uparrow\to\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f} \rangle$
and *?F* = $\langle set\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\} \rangle$
interpret ε: *is-cat-cone* α *E ?II* $\mathfrak{C}$ *?II-II* ε **by** (*rule assms*(*1*))
from *ε.cat-PL2-𝔣 ε.cat-PL2-𝔤* **have** *𝔤𝔣*: *?F* $\in_\circ$ *Vset* α
by (*intro Limit-vdoubleton-in-VsetI*) *auto*
from *assms*(*2,3*) **have**
$(\bigwedge \mathfrak{f}'.\ \mathfrak{f}' \in_\circ ?F \Longrightarrow (\lambda f \in_\circ ?F.\ (f = \mathfrak{f}_{PL}\ ?\ \mathfrak{f} : \mathfrak{g}))(\lfloor \mathfrak{f}' \rfloor) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b})$
by *auto*
note *cat-cone-cf-par-eps-NTMap-app* = *cat-cone-cf-par-eps-NTMap-app*
[
OF
assms(*1*)[
unfolded
the-cat-parallel-2-def the-cf-parallel-2-def $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*
],
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*, *OF - 𝔤𝔣 - this*,
simplified
]
from
cat-cone-cf-par-eps-NTMap-app[*of* $\mathfrak{g}_{PL}$, *simplified*]
cat-cone-cf-par-eps-NTMap-app[*of* $\mathfrak{f}_{PL}$, *simplified*]
cat-PL2-𝔤𝔣
show
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
by *fastforce+*

qed

lemma *cat-cocone-cf-par-2-eps-NTMap-app*:
assumes ε :
$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{b}_{PL2}\ \mathfrak{a}_{PL2}\ \mathfrak{f}_{PL}\ \mathfrak{g}_{PL}\ \mathfrak{b}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{g} >_{CF.cocone} E$:
$\uparrow\uparrow_{C}\ \mathfrak{b}_{PL2}\ \mathfrak{a}_{PL2}\ \mathfrak{f}_{PL}\ \mathfrak{g}_{PL} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
and $\mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
shows
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{g}$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{f}$
proof–
let *?II* = $\langle \uparrow\uparrow_{C}\ \mathfrak{b}_{PL2}\ \mathfrak{a}_{PL2}\ \mathfrak{f}_{PL}\ \mathfrak{g}_{PL} \rangle$
and *?II-II* = $\langle \uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{b}_{PL2}\ \mathfrak{a}_{PL2}\ \mathfrak{f}_{PL}\ \mathfrak{g}_{PL}\ \mathfrak{b}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{g} \rangle$
and *?F* = $\langle set\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\} \rangle$
have $\mathfrak{f}\mathfrak{g}$-$\mathfrak{g}\mathfrak{f}$: $\{\mathfrak{f}_{PL}, \mathfrak{g}_{PL}\} = \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **by** *auto*
interpret ε: *is-cat-cocone* α *E ?II* $\mathfrak{C}$ *?II-II* ε **by** (*rule assms*(*1*))
from ε.*cat-PL2-*$\mathfrak{f}$ ε.*cat-PL2-*$\mathfrak{g}$ **have** $\mathfrak{g}\mathfrak{f}$: *?F* $\in_{\circ}$ *Vset* α
by (*intro Limit-vdoubleton-in-VsetI*) *auto*
from *assms*(*2,3*) **have**
$(\bigwedge \mathfrak{f}'.\ \mathfrak{f}' \in_{\circ} ?F \Longrightarrow (\lambda f \in_{\circ} ?F.\ (f = \mathfrak{g}_{PL}\ ?\ \mathfrak{g} : \mathfrak{f}))(\lfloor \mathfrak{f}' \rfloor) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a})$
by *auto*
note *cat-cocone-cf-par-eps-NTMap-app* = *cat-cocone-cf-par-eps-NTMap-app*
[
OF assms(*1*)
[
unfolded
the-cat-parallel-2-def
the-cf-parallel-2-def
$\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*
insert-commute,
unfolded $\mathfrak{f}\mathfrak{g}$-$\mathfrak{g}\mathfrak{f}$
],
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*,
OF - $\mathfrak{g}\mathfrak{f}$ - *this*,
simplified
]
from
cat-cocone-cf-par-eps-NTMap-app[*of* $\mathfrak{g}_{PL}$, *simplified*]
cat-cocone-cf-par-eps-NTMap-app[*of* $\mathfrak{f}_{PL}$, *simplified*]
cat-PL2-$\mathfrak{g}\mathfrak{f}$
show
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{g}$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{f}$
by *fastforce+*
qed

lemma (**in** *is-cat-equalizer-2*) *cat-eq-2-eps-NTMap-app*:
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
proof–
have $\mathfrak{g}_{PL}$: $\mathfrak{g}_{PL} \in_{\circ} set\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **and** $\mathfrak{f}_{PL}$: $\mathfrak{f}_{PL} \in_{\circ} set\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **by** *auto*
note *cat-eq-eps-NTMap-app* = *is-cat-equalizer.cat-eq-eps-NTMap-app*
[
OF cat-equalizer-2-is-cat-equalizer,
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*
]
from *cat-eq-eps-NTMap-app*[*OF* $\mathfrak{g}_{PL}$] *cat-eq-eps-NTMap-app*[*OF* $\mathfrak{f}_{PL}$] *cat-PL2-*$\mathfrak{g}\mathfrak{f}$ **show**

$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
by *auto*
qed

lemma (**in** *is-cat-coequalizer-2*) *cat-coeq-2-eps-NTMap-app*:
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{g}$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{f}$
proof–
have $\mathfrak{g}_{PL}$: $\mathfrak{g}_{PL} \in_\circ$ *set* $\{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **and** $\mathfrak{f}_{PL}$: $\mathfrak{f}_{PL} \in_\circ$ *set* $\{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **by** *auto*
note *cat-eq-eps-NTMap-app* = *is-cat-coequalizer.cat-coeq-eps-NTMap-app*
[
OF cat-coequalizer-2-is-cat-coequalizer,
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*
]
from *cat-eq-eps-NTMap-app*[*OF* $\mathfrak{g}_{PL}$] *cat-eq-eps-NTMap-app*[*OF* $\mathfrak{f}_{PL}$] *cat-PL2-*$\mathfrak{gf}$ **show**
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{g}$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{b}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{f}$
by *auto*
qed

lemma (**in** *is-cat-equalizer-2*) *cat-eq-2-Comp-eq*:
$\mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) = \mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
$\mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) = \mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor)$
unfolding *cat-eq-2-eps-NTMap-app*[*symmetric*] **by** *simp-all*

lemma (**in** *is-cat-coequalizer-2*) *cat-coeq-2-Comp-eq*:
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{g} = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{f}$
$\varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{f} = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} \mathfrak{g}$
unfolding *cat-coeq-2-eps-NTMap-app*[*symmetric*] **by** *simp-all*

7.2.2 Universal property

lemma *is-cat-equalizer-2I′*:
assumes ε :
$E <_{CF.cone} \uparrow\uparrow\!\!\rightarrow\!\!\uparrow\uparrow_{CF} \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f} : \uparrow\uparrow_C \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
and $\mathfrak{f} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
and $\bigwedge\varepsilon'\ E'.\ \varepsilon'$:
$E' <_{CF.cone} \uparrow\uparrow\!\!\rightarrow\!\!\uparrow\uparrow_{CF} \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$:
$\uparrow\uparrow_C \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C} \Longrightarrow$
$\exists!f'.\ f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} f'$
shows $\varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
proof–
let *?II* = $\langle\uparrow\uparrow_C \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\rangle$
and *?II-II* = $\langle\uparrow\uparrow\!\!\rightarrow\!\!\uparrow\uparrow_{CF} \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\rangle$
and *?F* = $\langle$*set* $\{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}\rangle$
interpret ε: *is-cat-cone* α *E ?II* $\mathfrak{C}$ *?II-II* ε **by** (*rule assms*(*1*))
from ε*.cat-PL2-*$\mathfrak{f}$ ε*.cat-PL2-*$\mathfrak{g}$ **have** $\mathfrak{gf}$: *?F* $\in_\circ$ *Vset* α
by (*intro Limit-vdoubleton-in-VsetI*) *auto*
from *assms*(*2*,*3*) **have** $(\lambda f \in_\circ ?F.\ (f = \mathfrak{f}_{PL}\ ?\ \mathfrak{f} : \mathfrak{g}))(\lfloor \mathfrak{f}' \rfloor) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
if $\mathfrak{f}' \in_\circ$ *?F* **for** $\mathfrak{f}'$
using *that* **by** *simp*
note *is-cat-equalizerI′* = *is-cat-equalizerI′*
[
OF
assms(*1*)[
unfolded

```
        the-cat-parallel-2-def the-cf-parallel-2-def 𝔞_PL2-def 𝔟_PL2-def
      ],
    folded 𝔞_PL2-def 𝔟_PL2-def,
    OF
      -
      𝔤𝔣
      -
      this
      -
      assms(4)[unfolded the-cf-parallel-2-def the-cat-parallel-2-def],
    of 𝔤_PL,
    simplified
  ]
 show ?thesis by (rule cat-equalizer-is-cat-equalizer-2[OF is-cat-equalizerI'])
qed

lemma is-cat-coequalizer-2I':
 assumes ε :
  ↑↑→↑↑_CF ℭ 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL 𝔟 𝔞 𝔣 𝔤 >_CF.cocone E :
  ↑↑_C 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL ↦↦_Cα ℭ
  and 𝔤 : 𝔟 ↦_ℭ 𝔞
  and 𝔣 : 𝔟 ↦_ℭ 𝔞
  and ⋀ε' E'. ε' :
    ↑↑→↑↑_CF ℭ 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL 𝔟 𝔞 𝔣 𝔤 >_CF.cocone E' :
    ↑↑_C 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL ↦↦_Cα ℭ ⟹
    ∃!f'. f' : E ↦_ℭ E' ∧ ε'(⌊NTMap⌋)(⌊𝔞_PL2⌋) = f' ∘_Aℭ ε(⌊NTMap⌋)(⌊𝔞_PL2⌋)
 shows ε : (𝔞,𝔟,𝔤,𝔣) >_CF.coeq E : ↑↑_C ↦↦_Cα ℭ
proof–
 let ?II = ‹↑↑_C 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL›
  and ?II-II = ‹↑↑→↑↑_CF ℭ 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL 𝔟 𝔞 𝔣 𝔤›
  and ?F = ‹set {𝔤_PL, 𝔣_PL}›
 have 𝔣𝔤-𝔤𝔣: {𝔣_PL, 𝔤_PL} = {𝔤_PL, 𝔣_PL} by auto
 interpret ε: is-cat-cocone α E ?II ℭ ?II-II ε by (rule assms(1))
 from ε.cat-PL2-𝔣 ε.cat-PL2-𝔤 have 𝔤𝔣: ?F ∈_∘ Vset α
  by (intro Limit-vdoubleton-in-VsetI) auto
 from assms(2,3) have (λf∈_∘set {𝔤_PL, 𝔣_PL}. (f = 𝔤_PL ? 𝔤 : 𝔣))(⌊𝔣'⌋) : 𝔟 ↦_ℭ 𝔞
  if 𝔣' ∈_∘ set {𝔤_PL, 𝔣_PL} for 𝔣'
  using that by simp
 note is-cat-coequalizerI'
  [
    OF assms(1)[
      unfolded
        the-cat-parallel-2-def the-cf-parallel-2-def 𝔞_PL2-def 𝔟_PL2-def 𝔣𝔤-𝔤𝔣
        ],
    folded 𝔞_PL2-def 𝔟_PL2-def,
    OF
      -
      𝔤𝔣
      -
      this
      -
      assms(4)[unfolded the-cf-parallel-2-def the-cat-parallel-2-def 𝔣𝔤-𝔤𝔣],
    of 𝔤_PL,
    simplified
  ]
 with cat-PL2-𝔤𝔣 have
  ε : (𝔞,𝔟,?F,(λf∈_∘?F. (f = 𝔣_PL ? 𝔣 : 𝔤))) >_CF.coeq E : ⇑_C ↦↦_Cα ℭ
  by (auto simp: VLambda-vdoubleton)
```

from *cat-coequalizer-is-cat-coequalizer-2*[*OF this*] **show** *?thesis* **by** *simp*
qed

lemma (**in** *is-cat-equalizer-2*) *cat-eq-2-unique-cone*:
assumes ε' :
$E' <_{CF.cone} \uparrow\uparrow\to\uparrow\uparrow_{CF} \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$:
$\uparrow\uparrow_C \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists!f'.\ f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} f'$
by
(
rule is-cat-equalizer.cat-eq-unique-cone
[
OF cat-equalizer-2-is-cat-equalizer,
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*,
OF assms[*unfolded the-cf-parallel-2-def the-cat-parallel-2-def*]
]
)

lemma (**in** *is-cat-equalizer-2*) *cat-eq-2-unique*:
assumes $\varepsilon' : E' <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
$\exists!f'.\ f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon' = \varepsilon \cdot_{NTCF} ntcf\text{-}const\ (\uparrow\uparrow_C \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL})\ \mathfrak{C}\ f'$
proof–
interpret ε': *is-cat-equalizer-2* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{C}\ E'\ \varepsilon'$ **by** (*rule assms*)
show *?thesis*
by
(
rule is-cat-equalizer.cat-eq-unique
[
OF cat-equalizer-2-is-cat-equalizer,
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*,
OF ε'*.cat-equalizer-2-is-cat-equalizer*,
folded the-cat-parallel-2-def
]
)
qed

lemma (**in** *is-cat-equalizer-2*) *cat-eq-2-unique'*:
assumes $\varepsilon' : E' <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists!f'.\ f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) = \varepsilon(\lfloor NTMap \rfloor)(\lfloor \mathfrak{a}_{PL2} \rfloor) \circ_{A\mathfrak{C}} f'$
proof–
interpret ε': *is-cat-equalizer-2* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{C}\ E'\ \varepsilon'$ **by** (*rule assms*)
show *?thesis*
by
(
rule is-cat-equalizer.cat-eq-unique'
[
OF cat-equalizer-2-is-cat-equalizer,
folded $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def*,
OF ε'*.cat-equalizer-2-is-cat-equalizer*,
folded the-cat-parallel-2-def
]
)
qed

lemma (**in** *is-cat-coequalizer-2*) *cat-coeq-2-unique-cocone*:
assumes ε' :
$\uparrow\uparrow\to\uparrow\uparrow_{CF} \mathfrak{C}\ \mathfrak{b}_{PL2}\ \mathfrak{a}_{PL2}\ \mathfrak{f}_{PL}\ \mathfrak{g}_{PL}\ \mathfrak{b}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{g} >_{CF.cocone} E'$:

```
    ↑↑C 𝔟PL2 𝔞PL2 𝔣PL 𝔤PL ↦↦Cα ℭ
  shows ∃!f'. f' : E ↦ℭ E' ∧ ε'(⁅NTMap⁆)(⁅𝔞PL2⁆) = f' ∘Aℭ ε(⁅NTMap⁆)(⁅𝔞PL2⁆)
  by
    (
      rule is-cat-coequalizer.cat-coeq-unique-cocone
        [
          OF cat-coequalizer-2-is-cat-coequalizer,
          folded 𝔞PL2-def 𝔟PL2-def insert-commute,
          OF assms[
            unfolded
              the-cf-parallel-2-def the-cat-parallel-2-def cat-eq-F'-helper
            ]
        ]
    )

lemma (in is-cat-coequalizer-2) cat-coeq-2-unique:
  assumes ε' : (𝔞,𝔟,𝔤,𝔣) >CF.coeq E' : ↑↑C ↦↦Cα ℭ
  shows ∃!f'.
    f' : E ↦ℭ E' ∧
    ε' = ntcf-const (↑↑C 𝔟PL2 𝔞PL2 𝔣PL 𝔤PL) ℭ f' ∙NTCF ε
proof–
  interpret ε': is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E' ε' by (rule assms)
  show ?thesis
    by
      (
        rule is-cat-coequalizer.cat-coeq-unique
          [
            OF cat-coequalizer-2-is-cat-coequalizer,
            folded 𝔞PL2-def 𝔟PL2-def,
            OF ε'.cat-coequalizer-2-is-cat-coequalizer,
            folded the-cat-parallel-2-def the-cat-parallel-2-commute
          ]
      )
qed

lemma (in is-cat-coequalizer-2) cat-coeq-2-unique':
  assumes ε' : (𝔞,𝔟,𝔤,𝔣) >CF.coeq E' : ↑↑C ↦↦Cα ℭ
  shows ∃!f'. f' : E ↦ℭ E' ∧ ε'(⁅NTMap⁆)(⁅𝔞PL2⁆) = f' ∘Aℭ ε(⁅NTMap⁆)(⁅𝔞PL2⁆)
proof–
  interpret ε': is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E' ε' by (rule assms)
  show ?thesis
    by
      (
        rule is-cat-coequalizer.cat-coeq-unique'
          [
            OF cat-coequalizer-2-is-cat-coequalizer,
            folded 𝔞PL2-def 𝔟PL2-def,
            OF ε'.cat-coequalizer-2-is-cat-coequalizer,
            folded the-cat-parallel-2-def
          ]
      )
qed

lemma cat-equalizer-2-ex-is-iso-arr:
  assumes ε : E <CF.eq (𝔞,𝔟,𝔤,𝔣) : ↑↑C ↦↦Cα ℭ
    and ε' : E' <CF.eq (𝔞,𝔟,𝔤,𝔣) : ↑↑C ↦↦Cα ℭ
  obtains f where f : E' ↦iso ℭ E
    and ε' = ε ∙NTCF ntcf-const (↑↑C 𝔞PL2 𝔟PL2 𝔤PL 𝔣PL) ℭ f
```

```
proof–
  interpret ε: is-cat-equalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E ε by (rule assms(1))
  interpret ε′: is-cat-equalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E′ ε′ by (rule assms(2))
  show ?thesis
    using that
    by
      (
        rule cat-equalizer-ex-is-iso-arr
          [
            OF
              ε.cat-equalizer-2-is-cat-equalizer
              ε′.cat-equalizer-2-is-cat-equalizer,
            folded 𝔞_PL2-def 𝔟_PL2-def the-cat-parallel-2-def
          ]
      )
qed

lemma cat-equalizer-2-ex-is-iso-arr′:
  assumes ε : E <_CF.eq (𝔞,𝔟,𝔤,𝔣) : ↑↑_C ↦↦_Cα ℭ
    and ε′ : E′ <_CF.eq (𝔞,𝔟,𝔤,𝔣) : ↑↑_C ↦↦_Cα ℭ
  obtains f where f : E′ ↦_iso ℭ E
    and ε′(|NTMap|)(|𝔞_PL2|) = ε(|NTMap|)(|𝔞_PL2|) ∘_A ℭ f
    and ε′(|NTMap|)(|𝔟_PL2|) = ε(|NTMap|)(|𝔟_PL2|) ∘_A ℭ f
proof–
  interpret ε: is-cat-equalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E ε by (rule assms(1))
  interpret ε′: is-cat-equalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E′ ε′ by (rule assms(2))
  show ?thesis
    using that
    by
      (
        rule cat-equalizer-ex-is-iso-arr′
          [
            OF
              ε.cat-equalizer-2-is-cat-equalizer
              ε′.cat-equalizer-2-is-cat-equalizer,
            folded 𝔞_PL2-def 𝔟_PL2-def the-cat-parallel-2-def
          ]
      )
qed

lemma cat-coequalizer-2-ex-is-iso-arr:
  assumes ε : (𝔞,𝔟,𝔤,𝔣) >_CF.coeq E : ↑↑_C ↦↦_Cα ℭ
    and ε′ : (𝔞,𝔟,𝔤,𝔣) >_CF.coeq E′ : ↑↑_C ↦↦_Cα ℭ
  obtains f where f : E ↦_iso ℭ E′
    and ε′ = ntcf-const (↑↑_C 𝔟_PL2 𝔞_PL2 𝔣_PL 𝔤_PL) ℭ f ⊙_NTCF ε
proof–
  interpret ε: is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E ε by (rule assms(1))
  interpret ε′: is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E′ ε′ by (rule assms(2))
  show ?thesis
    using that
    by
      (
        rule cat-coequalizer-ex-is-iso-arr
          [
            OF
              ε.cat-coequalizer-2-is-cat-coequalizer
              ε′.cat-coequalizer-2-is-cat-coequalizer,
            folded
```

```
      𝔞_PL2-def 𝔟_PL2-def the-cat-parallel-2-def the-cat-parallel-2-commute
    ]
  )
qed

lemma cat-coequalizer-2-ex-is-iso-arr':
  assumes ε : (𝔞,𝔟,𝔤,𝔣) >_CF.coeq E : ↑↑_C ↦↦_Cα ℭ
    and ε' : (𝔞,𝔟,𝔤,𝔣) >_CF.coeq E' : ↑↑_C ↦↦_Cα ℭ
  obtains f where f : E ↦_iso ℭ E'
    and ε'(|NTMap|)(|𝔞_PL2|) = f ∘_A ℭ ε(|NTMap|)(|𝔞_PL2|)
    and ε'(|NTMap|)(|𝔟_PL2|) = f ∘_A ℭ ε(|NTMap|)(|𝔟_PL2|)
proof–
  interpret ε: is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E ε by (rule assms(1))
  interpret ε': is-cat-coequalizer-2 α 𝔞 𝔟 𝔤 𝔣 ℭ E' ε' by (rule assms(2))
  show ?thesis
    using that
    by
      (
        rule cat-coequalizer-ex-is-iso-arr'
          [
            OF
              ε.cat-coequalizer-2-is-cat-coequalizer
              ε'.cat-coequalizer-2-is-cat-coequalizer,
            folded
              𝔞_PL2-def 𝔟_PL2-def the-cat-parallel-2-def the-cat-parallel-2-commute
          ]
      )
qed
```

7.2.3 Further properties

```
lemma (in is-cat-equalizer-2) cat-eq-2-is-monic-arr:
  ε(|NTMap|)(|𝔞_PL2|) : E ↦_mon ℭ 𝔞
  by
    (
      rule is-cat-equalizer.cat-eq-is-monic-arr[
        OF cat-equalizer-2-is-cat-equalizer, folded 𝔞_PL2-def
        ]
    )

lemma (in is-cat-coequalizer-2) cat-coeq-2-is-epic-arr:
  ε(|NTMap|)(|𝔞_PL2|) : 𝔞 ↦_epi ℭ E
  by
    (
      rule is-cat-coequalizer.cat-coeq-is-epic-arr[
        OF cat-coequalizer-2-is-cat-coequalizer, folded 𝔞_PL2-def
        ]
    )
```

7.3 Equalizer cone

7.3.1 Definition and elementary properties

```
definition ntcf-equalizer-base :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ (V ⇒ V) ⇒ V
  where ntcf-equalizer-base ℭ 𝔞 𝔟 𝔤 𝔣 E e =
    [
      (λx∈₀↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL(|Obj|). e x),
      cf-const (↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL) ℭ E,
      ↑↑→↑↑_CF ℭ 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL 𝔞 𝔟 𝔤 𝔣,
```

$\uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}$,
$\mathfrak{C}$
]$_\circ$

Components.

lemma *ntcf-equalizer-base-components*:
shows *ntcf-equalizer-base* $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e(\lfloor NTMap \rfloor) =$
$(\lambda x \in_\circ \uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}(\lfloor Obj \rfloor).\ e\ x)$
and [*cat-lim-cs-simps*]: *ntcf-equalizer-base* $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e(\lfloor NTDom \rfloor) =$
cf-const $(\uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL})\ \mathfrak{C}\ E$
and [*cat-lim-cs-simps*]: *ntcf-equalizer-base* $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e(\lfloor NTCod \rfloor) =$
$\uparrow\uparrow\!\!\rightarrow\!\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$
and [*cat-lim-cs-simps*]:
ntcf-equalizer-base $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e(\lfloor NTDGDom \rfloor) = \uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}$
and [*cat-lim-cs-simps*]:
ntcf-equalizer-base $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e(\lfloor NTDGCod \rfloor) = \mathfrak{C}$
unfolding *ntcf-equalizer-base-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

7.3.2 Natural transformation map

mk-VLambda *ntcf-equalizer-base-components*(*1*)
|*vsv ntcf-equalizer-base-NTMap-vsv*[*cat-lim-cs-intros*]|
|*vdomain ntcf-equalizer-base-NTMap-vdomain*[*cat-lim-cs-simps*]|
|*app ntcf-equalizer-base-NTMap-app*[*cat-lim-cs-simps*]|

7.3.3 Equalizer cone is a cone

lemma (**in** *category*) *cat-ntcf-equalizer-base-is-cat-cone*:
assumes $e\ \mathfrak{a}_{PL2} : E \mapsto_{\mathfrak{C}} \mathfrak{a}$
and $e\ \mathfrak{b}_{PL2} : E \mapsto_{\mathfrak{C}} \mathfrak{b}$
and $e\ \mathfrak{b}_{PL2} = \mathfrak{g} \circ_{A\mathfrak{C}} e\ \mathfrak{a}_{PL2}$
and $e\ \mathfrak{b}_{PL2} = \mathfrak{f} \circ_{A\mathfrak{C}} e\ \mathfrak{a}_{PL2}$
and $\mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
and $\mathfrak{f} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$
shows *ntcf-equalizer-base* $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e$:
$E <_{CF.cone} \uparrow\uparrow\!\!\rightarrow\!\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$:
$\uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret *par*: *cf-parallel-2* $\alpha\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{C}$
by (*intro cf-parallel-2I cat-parallel-2I assms*(*5*,*6*))
(*simp-all add*: *cat-parallel-cs-intros cat-cs-intros*)
show *?thesis*
proof(*intro is-cat-coneI is-tm-ntcfI′ is-ntcfI′*)
show *vfsequence* (*ntcf-equalizer-base* $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e$)
unfolding *ntcf-equalizer-base-def* **by** *auto*
show *vcard* (*ntcf-equalizer-base* $\mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ E\ e$) = $5_{\mathbb{N}}$
unfolding *ntcf-equalizer-base-def* **by** (*simp add*: *nat-omega-simps*)
from *assms*(*2*) **show**
cf-const $(\uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL})\ \mathfrak{C}\ E : \uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-parallel-cs-intros cat-cs-intros*
)
from *assms* **show**
$\uparrow\uparrow\!\!\rightarrow\!\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f} : \uparrow\uparrow_C\ \mathfrak{a}_{PL2}\ \mathfrak{b}_{PL2}\ \mathfrak{g}_{PL}\ \mathfrak{f}_{PL} \mapsto\!\mapsto_{C\alpha} \mathfrak{C}$

```
    by (cs-concl cs-intro: cat-parallel-cs-intros cat-small-cs-intros)
  show
    ntcf-equalizer-base ℭ 𝔞 𝔟 𝔤 𝔣 E e(|NTMap|)(|i|) :
      cf-const (↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL) ℭ E(|ObjMap|)(|i|) ↦_ℭ
      ↑↑→↑↑_CF ℭ 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL 𝔞 𝔟 𝔤 𝔣(|ObjMap|)(|i|)
    if i ∈_∘ ↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL(|Obj|) for i
  proof–
    from that assms(1,2,5,6) show ?thesis
      by (elim the-cat-parallel-2-ObjE; simp only:)
        (
          cs-concl
            cs-simp: cat-lim-cs-simps cat-cs-simps cat-parallel-cs-simps
            cs-intro: cat-cs-intros cat-parallel-cs-intros
        )
  qed
  show
    ntcf-equalizer-base ℭ 𝔞 𝔟 𝔤 𝔣 E e(|NTMap|)(|b'|) ∘_Aℭ
      cf-const (↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL) ℭ E(|ArrMap|)(|f'|) =
        ↑↑→↑↑_CF ℭ 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL 𝔞 𝔟 𝔤 𝔣(|ArrMap|)(|f'|) ∘_Aℭ
        ntcf-equalizer-base ℭ 𝔞 𝔟 𝔤 𝔣 E e(|NTMap|)(|a'|)
    if f' : a' ↦_↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL b' for a' b' f'
    using that assms(1,2,5,6)
    by (elim par.the-cat-parallel-2-is-arrE; simp only:)
      (
        cs-concl
          cs-simp:
            cat-cs-simps
            cat-lim-cs-simps
            cat-parallel-cs-simps
            assms(3,4)[symmetric]
          cs-intro: cat-parallel-cs-intros
      )+
qed
  (
    use assms(2) in
      ‹
        cs-concl
          cs-intro: cat-lim-cs-intros cat-cs-intros
          cs-simp: cat-lim-cs-simps
      ›
  )+
qed
```

8 Pointed arrows and natural transformations

8.1 Pointed arrow

The terminology that is used in this section deviates from convention: a pointed arrow is merely an arrow in *Set* from a singleton set to another set.

8.1.1 Definition and elementary properties

See Chapter III-2 in [9].

definition *ntcf-paa* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-paa* $\mathfrak{a}$ B b = $[(\lambda a \in_{\circ} set\ \{\mathfrak{a}\}.\ b),\ set\ \{\mathfrak{a}\},\ B]_{\circ}$

Components.

lemma *ntcf-paa-components*:
  shows *ntcf-paa* $\mathfrak{a}$ B $b(\!|ArrVal|\!)$ = $(\lambda a \in_{\circ} set\ \{\mathfrak{a}\}.\ b)$
    and [*cat-cs-simps*]: *ntcf-paa* $\mathfrak{a}$ B $b(\!|ArrDom|\!)$ = *set* $\{\mathfrak{a}\}$
    and [*cat-cs-simps*]: *ntcf-paa* $\mathfrak{a}$ B $b(\!|ArrCod|\!)$ = B
  unfolding *ntcf-paa-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

8.1.2 Arrow value

mk-VLambda *ntcf-paa-components*(*1*)
  |*vsv ntcf-paa-ArrVal-vsv*[*cat-cs-intros*]|
  |*vdomain ntcf-paa-ArrVal-vdomain*[*cat-cs-simps*]|
  |*app ntcf-paa-ArrVal-app*[*unfolded vsingleton-iff*, *cat-cs-simps*]|

8.1.3 Pointed arrow is an arrow in *Set*

lemma (**in** $\mathcal{Z}$) *ntcf-paa-is-arr*:
  assumes $\mathfrak{a} \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $A \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $a \in_{\circ} A$
  shows *ntcf-paa* $\mathfrak{a}$ A a : *set* $\{\mathfrak{a}\} \mapsto_{cat\text{-}Set\ \alpha} A$
proof(*intro cat-Set-is-arrI arr-SetI cat-cs-intros*, *unfold cat-cs-simps*)
  show *vfsequence* (*ntcf-paa* $\mathfrak{a}$ A a) **unfolding** *ntcf-paa-def* **by** *simp*
  show *vcard* (*ntcf-paa* $\mathfrak{a}$ A a) = $3_{\mathbb{N}}$
    unfolding *ntcf-paa-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{R}_{\circ}$ (*ntcf-paa* $\mathfrak{a}$ A $a(\!|ArrVal|\!)$) $\subseteq_{\circ} A$
    unfolding *ntcf-paa-components* **by** (*intro vrange-VLambda-vsubset assms*)
qed (*use assms* **in** ‹*auto simp*: *cat-Set-components*(*1*) *Limit-vsingleton-in-VsetI*›)

lemma (**in** $\mathcal{Z}$) *ntcf-paa-is-arr'*[*cat-cs-intros*]:
  assumes $\mathfrak{a} \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
    and $A \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
    and $a \in_{\circ} A$
    and $A' = set\ \{\mathfrak{a}\}$
    and $B' = A$
    and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
  shows *ntcf-paa* $\mathfrak{a}$ A a : $A' \mapsto_{\mathfrak{C}'} B'$
  using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule ntcf-paa-is-arr*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$.*ntcf-paa-is-arr'*

8.1.4 Further properties

lemma *ntcf-paa-injective*[*cat-cs-simps*]:
  ntcf-paa $\mathfrak{a}$ A b = *ntcf-paa* $\mathfrak{a}$ A $c \longleftrightarrow b = c$
proof

```
  assume ntcf-paa 𝔞 A b = ntcf-paa 𝔞 A c
  then have ntcf-paa 𝔞 A b(|ArrVal|)(|𝔞|) = ntcf-paa 𝔞 A c(|ArrVal|)(|𝔞|) by simp
  then show b = c by (cs-prems cs-simp: cat-cs-simps)
qed simp

lemma (in 𝒵) ntcf-paa-ArrVal:
  assumes F : set {𝔞} ↦_cat-Set α X
  shows ntcf-paa 𝔞 X (F(|ArrVal|)(|𝔞|)) = F
proof–
  interpret F: arr-Set α F
    rewrites [cat-cs-simps]: F(|ArrDom|) = set {𝔞}
      and [cat-cs-simps]: F(|ArrCod|) = X
    by (auto simp: cat-Set-is-arrD[OF assms])
  from F.arr-Par-ArrDom-in-Vset have 𝔞: 𝔞 ∈₀ Vset α by auto
  from assms 𝔞 F.arr-Par-ArrCod-in-Vset have lhs-is-arr:
    ntcf-paa 𝔞 X (F(|ArrVal|)(|𝔞|)) : set {𝔞} ↦_cat-Set α X
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Set-components(1)
          cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
      )
  then have dom-lhs: 𝒟₀ (ntcf-paa 𝔞 X (F(|ArrVal|)(|𝔞|))(|ArrVal|)) = set {𝔞}
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms have dom-rhs: 𝒟₀ (F(|ArrVal|)) = set {𝔞}
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
  proof(rule arr-Set-eqI)
    from lhs-is-arr assms
    show arr-Set-lhs: arr-Set α (ntcf-paa 𝔞 X (F(|ArrVal|)(|𝔞|)))
      and arr-Set-rhs: arr-Set α F
      by (auto dest: cat-Set-is-arrD)
    show ntcf-paa 𝔞 X (F(|ArrVal|)(|𝔞|))(|ArrVal|) = F(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs vsingleton-iff; (simp only:)?)
      show ntcf-paa 𝔞 X (F(|ArrVal|)(|𝔞|))(|ArrVal|)(|𝔞|) = F(|ArrVal|)(|𝔞|)
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed (use assms in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
qed

lemma (in 𝒵) ntcf-paa-ArrVal':
  assumes F : set {𝔞} ↦_cat-Set α X and a = 𝔞
  shows ntcf-paa 𝔞 X (F(|ArrVal|)(|a|)) = F
  using assms(1) unfolding assms(2) by (rule ntcf-paa-ArrVal)

lemma (in 𝒵) ntcf-paa-Comp-right[cat-cs-simps]:
  assumes F : A ↦_cat-Set α B
    and 𝔞 ∈₀ cat-Set α(|Obj|)
    and a ∈₀ A
  shows F ∘_A cat-Set α ntcf-paa 𝔞 A a = ntcf-paa 𝔞 B (F(|ArrVal|)(|a|))
proof–
  from assms have F-paa:
    F ∘_A cat-Set α ntcf-paa 𝔞 A a : set {𝔞} ↦_cat-Set α B
    by (cs-concl cs-intro: cat-cs-intros)
  then have dom-lhs: 𝒟₀ ((F ∘_A cat-Set α ntcf-paa 𝔞 A a)(|ArrVal|)) = set {𝔞}
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms have paa: ntcf-paa 𝔞 B (F(|ArrVal|)(|a|)) : set {𝔞} ↦_cat-Set α B
    by (cs-concl cs-shallow cs-intro: cat-Set-cs-intros cat-cs-intros)
```

```
  then have dom-rhs: 𝒟∘ ((ntcf-paa 𝔞 B (F(|ArrVal|)(|a|)))(|ArrVal|)) = set {𝔞}
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
  proof(rule arr-Set-eqI)
    from F-paa paa assms
    show arr-Set α (F ∘_A cat-Set α ntcf-paa 𝔞 A a)
      and arr-Set-rhs: arr-Set α (ntcf-paa 𝔞 B (F(|ArrVal|)(|a|)))
      by (auto dest: cat-Set-is-arrD)
    show
      (F ∘_A cat-Set α ntcf-paa 𝔞 A a)(|ArrVal|) =
        ntcf-paa 𝔞 B (F(|ArrVal|)(|a|))(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs vsingleton-iff; (simp only:)?)
      from assms show
        (F ∘_A cat-Set α ntcf-paa 𝔞 A a)(|ArrVal|)(|𝔞|) =
          ntcf-paa 𝔞 B (F(|ArrVal|)(|a|))(|ArrVal|)(|𝔞|)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros)
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed (use F-paa paa in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
qed

lemmas [cat-cs-simps] = 𝒵.ntcf-paa-Comp-right
```

8.2 Pointed natural transformation

8.2.1 Definition and elementary properties

See Chapter III-2 in [9].

```
definition ntcf-pointed :: V ⇒ V ⇒ V
  where ntcf-pointed α 𝔞 =
    [
      (
        λx∈∘cat-Set α(|Obj|).
          [
            (λf∈∘Hom (cat-Set α) (set {𝔞}) x. f(|ArrVal|)(|𝔞|)),
            Hom (cat-Set α) (set {𝔞}) x,
            x
          ]∘
      ),
      Hom_{O.C α} cat-Set α(set {𝔞},−),
      cf-id (cat-Set α),
      cat-Set α,
      cat-Set α
    ]∘
```

Components.

```
lemma ntcf-pointed-components:
  shows ntcf-pointed α 𝔞(|NTMap|) =
      (
        λx∈∘cat-Set α(|Obj|).
          [
            (λf∈∘Hom (cat-Set α) (set {𝔞}) x. f(|ArrVal|)(|𝔞|)),
            Hom (cat-Set α) (set {𝔞}) x,
            x
          ]∘
      )
    and [cat-cs-simps]: ntcf-pointed α 𝔞(|NTDom|) = Hom_{O.C α} cat-Set α(set {𝔞},−)
    and [cat-cs-simps]: ntcf-pointed α 𝔞(|NTCod|) = cf-id (cat-Set α)
    and [cat-cs-simps]: ntcf-pointed α 𝔞(|NTDGDom|) = cat-Set α
```

and [*cat-cs-simps*]: *ntcf-pointed* α $\mathfrak{a}(\!|NTDGCod|\!)$ = *cat-Set* α
unfolding *ntcf-pointed-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

8.2.2 Natural transformation map

mk-VLambda *ntcf-pointed-components*(*1*)
|*vsv ntcf-pointed-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-pointed-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-pointed-NTMap-app'*|

lemma (**in** $\mathcal{Z}$) *ntcf-pointed-NTMap-app-ArrVal-app*[*cat-cs-simps*]:
assumes $X \in_\circ$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** F : *set* $\{\mathfrak{a}\} \mapsto_{cat\text{-}Set\ \alpha}$ X
shows *ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)(\!|F|\!) = F(\!|ArrVal|\!)(\!|\mathfrak{a}|\!)$
by (*simp add*: *assms*(*2*) *ntcf-pointed-NTMap-app'*[*OF assms*(*1*)] *arr-Rel-components*)

lemma (**in** $\mathcal{Z}$) *ntcf-pointed-NTMap-app-is-iso-arr*:
assumes $\mathfrak{a} \in_\circ$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $X \in_\circ$ *cat-Set* $\alpha(\!|Obj|\!)$
shows *ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)$:
Hom (*cat-Set* α) (*set* $\{\mathfrak{a}\}$) $X \mapsto_{iso\, cat\text{-}Set\ \alpha}$ X
proof–
interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)
note *app-X* = *ntcf-pointed-NTMap-app'*[*OF assms*(*2*)]
show *?thesis*
proof(*intro cat-Set-is-iso-arrI cat-Set-is-arrI arr-SetI*)
show *ArrVal-vsv*: *vsv* (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$)
unfolding *app-X arr-Rel-components* **by** *simp*
show *vcard* (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)$) = $3_{\mathbb{N}}$
unfolding *app-X arr-Rel-components* **by** (*simp add*: *nat-omega-simps*)
show *ArrVal-vdomain*:
$\mathcal{D}_\circ$ (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$) = *Hom* (*cat-Set* α) (*set* $\{\mathfrak{a}\}$) X
unfolding *app-X arr-Rel-components* **by** *simp*
show *vrange-left*:
$\mathcal{R}_\circ$ (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$) $\subseteq_\circ$
ntcf-pointed α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrCod|\!)$
unfolding *app-X arr-Rel-components*
by
(
auto
simp: *in-Hom-iff*
intro: *cat-Set-cs-intros*
intro!: *vrange-VLambda-vsubset*
)
show $\mathcal{R}_\circ$ (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$) = X
proof(*intro vsubset-antisym*)
show $X \subseteq_\circ \mathcal{R}_\circ$ (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$)
proof(*intro vsubsetI*)
fix x **assume** *prems*: $x \in_\circ X$
from *assms prems* **have** *F-in-vdomain*:
ntcf-paa $\mathfrak{a}$ X $x \in_\circ \mathcal{D}_\circ$ ((*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$))
unfolding *app-X arr-Rel-components vdomain-VLambda in-Hom-iff*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms prems* **have** *x-def*:
x = *ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)(\!|$*ntcf-paa* $\mathfrak{a}$ X $x|\!|)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $x \in_\circ \mathcal{R}_\circ$ (*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$)
by (*subst x-def*) (*intro vsv.vsv-vimageI2 F-in-vdomain ArrVal-vsv*)
qed
qed (*use vrange-left* **in** ‹*simp add*: *app-X arr-Rel-components*›)

```
    from assms show ntcf-pointed α 𝔞(|NTMap|)(|X|)(|ArrDom|) ∈∘ Vset α
      unfolding app-X arr-Rel-components cat-Set-components(1)
      by (intro Set.cat-Hom-in-Vset[OF - assms(2)])
        (auto simp: cat-Set-components(1))
    show v11 (ntcf-pointed α 𝔞(|NTMap|)(|X|)(|ArrVal|))
    proof(intro vsv.vsv-valeq-v11I ArrVal-vsv, unfold ArrVal-vdomain in-Hom-iff)
      fix F G assume prems:
        F : set {𝔞} ↦cat-Set α X
        G : set {𝔞} ↦cat-Set α X
        ntcf-pointed α 𝔞(|NTMap|)(|X|)(|ArrVal|)(|F|) =
          ntcf-pointed α 𝔞(|NTMap|)(|X|)(|ArrVal|)(|G|)
      note F = cat-Set-is-arrD[OF prems(1)] and G = cat-Set-is-arrD[OF prems(2)]
      from prems(3,1,2) assms have F-ArrVal-G-ArrVal: F(|ArrVal|)(|𝔞|) = G(|ArrVal|)(|𝔞|)
        by (cs-prems cs-simp: cat-cs-simps)
      interpret F: arr-Set α F + G: arr-Set α G by (simp-all add: F G)
      show F = G
      proof(rule arr-Set-eqI)
        show arr-Set α F arr-Set α G
          by (intro F.arr-Set-axioms G.arr-Set-axioms)+
        show F(|ArrVal|) = G(|ArrVal|)
          by
            (
              rule vsv-eqI,
              unfold F.arr-Set-ArrVal-vdomain G.arr-Set-ArrVal-vdomain F(2) G(2)
            )
            (auto simp: F-ArrVal-G-ArrVal)
      qed (simp-all add: F G)
    qed
  qed (use assms in ‹auto simp: app-X arr-Rel-components cat-Set-components(1)›)
qed

lemma (in 𝒵) ntcf-pointed-NTMap-app-is-iso-arr'[cat-cs-intros]:
  assumes 𝔞 ∈∘ cat-Set α(|Obj|)
    and X ∈∘ cat-Set α(|Obj|)
    and A' = Hom (cat-Set α) (set {𝔞}) X
    and B' = X
    and 𝔇' = cat-Set α
  shows ntcf-pointed α 𝔞(|NTMap|)(|X|) : A' ↦iso𝔇' B'
  using assms(1,2)
  unfolding assms(3–5)
  by (rule ntcf-pointed-NTMap-app-is-iso-arr)

lemmas [cat-cs-intros] = 𝒵.ntcf-pointed-NTMap-app-is-iso-arr'

lemmas (in 𝒵) ntcf-pointed-NTMap-app-is-arr'[cat-cs-intros] =
  is-iso-arrD(1)[OF 𝒵.ntcf-pointed-NTMap-app-is-iso-arr']

lemmas [cat-cs-intros] = 𝒵.ntcf-pointed-NTMap-app-is-arr'
```

8.2.3 Pointed natural transformation is a natural isomorphism

```
lemma (in 𝒵) ntcf-pointed-is-iso-ntcf:
  assumes 𝔞 ∈∘ cat-Set α(|Obj|)
  shows ntcf-pointed α 𝔞 :
    Hom_{O.Cα} cat-Set α(set {𝔞},−) ↦_{CF.iso} cf-id (cat-Set α) :
    cat-Set α ↦↦_{Cα} cat-Set α
proof(intro is-iso-ntcfI is-ntcfI')
```

```
note 𝔞 = assms[unfolded cat-Set-components(1)]
from assms have set-𝔞: set {𝔞} ∈∘ cat-Set α(|Obj|)
  unfolding cat-Set-components by auto

show vfsequence (ntcf-pointed α 𝔞) unfolding ntcf-pointed-def by auto
show vcard (ntcf-pointed α 𝔞) = 5ℕ
  unfolding ntcf-pointed-def by (auto simp: nat-omega-simps)
from assms set-𝔞 show
  Hom_{O.Cα} cat-Set α(set {𝔞},−) : cat-Set α ↦↦_{Cα} cat-Set α
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
show ntcf-pointed α 𝔞(|NTMap|)(|a|) :
  Hom_{O.Cα} cat-Set α(set {𝔞},−)(|ObjMap|)(|a|) ↦_{cat-Set α}
  cf-id (cat-Set α)(|ObjMap|)(|a|)
  if a ∈∘ cat-Set α(|Obj|) for a
  using assms that set-𝔞
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
    )
show
  ntcf-pointed α 𝔞(|NTMap|)(|b|) ∘_{A cat-Set α}
    Hom_{O.Cα} cat-Set α (set {𝔞},−)(|ArrMap|)(|f|) =
      cf-id (cat-Set α)(|ArrMap|)(|f|) ∘_{A cat-Set α} ntcf-pointed α 𝔞(|NTMap|)(|a|)
  if f : a ↦_{cat-Set α} b for a b f
proof–
  let ?pb = ‹ntcf-pointed α 𝔞(|NTMap|)(|b|)›
    and ?pa = ‹ntcf-pointed α 𝔞(|NTMap|)(|a|)›
    and ?hom = ‹cf-hom (cat-Set α) [cat-Set α(|CId|)(|set {𝔞}|), f]∘›
  from assms set-𝔞 that have pb-hom:
    ?pb ∘_{A cat-Set α} ?hom : Hom (cat-Set α) (set {𝔞}) a ↦_{cat-Set α} b
    by
      (
        cs-concl cs-shallow
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  then have dom-lhs:
    𝒟∘ ((?pb ∘_{A cat-Set α} ?hom)(|ArrVal|)) = Hom (cat-Set α) (set {𝔞}) a
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms set-𝔞 that have f-pa:
    f ∘_{A cat-Set α} ?pa : Hom (cat-Set α) (set {𝔞}) a ↦_{cat-Set α} b
    by (cs-concl cs-intro: cat-cs-intros)
  then have dom-rhs:
    𝒟∘ ((f ∘_{A cat-Set α} ?pa)(|ArrVal|)) = Hom (cat-Set α) (set {𝔞}) a
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have [cat-cs-simps]: ?pb ∘_{A cat-Set α} ?hom = f ∘_{A cat-Set α} ?pa
  proof(rule arr-Set-eqI)
    from pb-hom show arr-Set α (?pb ∘_{A cat-Set α} ?hom)
      by (auto dest: cat-Set-is-arrD(1))
    from f-pa show arr-Set α (f ∘_{A cat-Set α} ?pa)
      by (auto dest: cat-Set-is-arrD(1))
    show (?pb ∘_{A cat-Set α} ?hom)(|ArrVal|) = (f ∘_{A cat-Set α} ?pa)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix g assume g : set {𝔞} ↦_{cat-Set α} a
      with assms 𝔞 set-𝔞 that show
        (?pb ∘_{A cat-Set α} ?hom)(|ArrVal|)(|g|) = (f ∘_{A cat-Set α} ?pa)(|ArrVal|)(|g|)
        by
          (
```

```
            cs-concl cs-shallow
              cs-simp: V-cs-simps cat-cs-simps
              cs-intro:
                V-cs-intros cat-cs-intros cat-op-intros cat-prod-cs-intros
          )
      qed (use arr-Set-pb-hom arr-Set-f-pa in auto)
    qed (use pb-hom f-pa in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
    from assms that set-𝔞 show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
        )
  qed
  show ntcf-pointed α 𝔞(|NTMap|)(|a|) :
    Hom_{O.CA}cat-Set α(set {𝔞},−)(|ObjMap|)(|a|) ↦_{iso}cat-Set α
    cf-id (cat-Set α)(|ObjMap|)(|a|)
    if a ∈₀ cat-Set α(|Obj|) for a
    using assms 𝔞 set-𝔞 that
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
      )

qed (auto simp: ntcf-pointed-components intro: cat-cs-intros)

lemma (in 𝒵) ntcf-pointed-is-iso-ntcf'[cat-cs-intros]:
  assumes 𝔞 ∈₀ cat-Set α(|Obj|)
    and 𝔉' = Hom_{O.CA}cat-Set α(set {𝔞},−)
    and 𝔊' = cf-id (cat-Set α)
    and 𝔄' = cat-Set α
    and 𝔅' = cat-Set α
    and α' = α
  shows ntcf-pointed α 𝔞 : 𝔉' ↦_{CF.iso} 𝔊' : 𝔄' ↦↦_{C_α'} 𝔅'
  using assms(1) unfolding assms(2–6) by (rule ntcf-pointed-is-iso-ntcf)

lemmas [cat-cs-intros] = 𝒵.ntcf-pointed-is-iso-ntcf'
```

8.3 Inverse pointed natural transformation

8.3.1 Definition and elementary properties

See Chapter III-2 in [9].

```
definition ntcf-pointed-inv :: V ⇒ V ⇒ V
  where ntcf-pointed-inv α 𝔞 =
    [
      (
        λX∈₀cat-Set α(|Obj|).
          [(λx∈₀X. ntcf-paa 𝔞 X x), X, Hom (cat-Set α) (set {𝔞}) X]₀
      ),
      cf-id (cat-Set α),
      Hom_{O.CA}cat-Set α(set {𝔞},−),
      cat-Set α,
      cat-Set α
    ]₀
```

Components.

lemma *ntcf-pointed-inv-components*:
shows *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)$ =
(
$\lambda X \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!).$
$[(\lambda x \in_{\circ} X.\ ntcf\text{-}paa\ \mathfrak{a}\ X\ x),\ X,\ Hom\ (cat\text{-}Set\ \alpha)\ (set\ \{\mathfrak{a}\})\ X]_{\circ}$
)
and [*cat-cs-simps*]: *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTDom|\!)$ = *cf-id* (*cat-Set* α)
and [*cat-cs-simps*]:
ntcf-pointed-inv α $\mathfrak{a}(\!|NTCod|\!)$ = $Hom_{O.C\alpha}$ *cat-Set* α(*set* $\{\mathfrak{a}\}$,−)
and [*cat-cs-simps*]: *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTDGDom|\!)$ = *cat-Set* α
and [*cat-cs-simps*]: *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTDGCod|\!)$ = *cat-Set* α
unfolding *ntcf-pointed-inv-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

8.3.2 Natural transformation map

mk-VLambda *ntcf-pointed-inv-components*(*1*)
|*vsv ntcf-pointed-inv-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-pointed-inv-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-pointed-inv-NTMap-app′*|

lemma (**in** $\mathcal{Z}$) *ntcf-pointed-inv-NTMap-app-ArrVal-app*[*cat-cs-simps*]:
assumes $X \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $x \in_{\circ} X$
shows *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)(\!|x|\!)$ = *ntcf-paa* $\mathfrak{a}$ X x
by
(
simp add:
assms(*2*) *ntcf-pointed-inv-NTMap-app′*[*OF assms*(*1*)] *arr-Rel-components*
)

lemma (**in** $\mathcal{Z}$) *ntcf-pointed-inv-NTMap-app-is-arr*:
assumes $\mathfrak{a} \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $X \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
shows *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)$:
$X \mapsto_{cat\text{-}Set\ \alpha} Hom\ (cat\text{-}Set\ \alpha)\ (set\ \{\mathfrak{a}\})\ X$
proof–
interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)
note *app-X* = *ntcf-pointed-inv-NTMap-app′*[*OF assms*(*2*)]
show *?thesis*
proof(*intro cat-Set-is-arrI arr-SetI*)
show *ArrVal-vsv*: *vsv* (*ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$)
unfolding *app-X arr-Rel-components* **by** *simp*
show *vcard* (*ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)$) = $3_{\mathbb{N}}$
unfolding *app-X arr-Rel-components* **by** (*simp add*: *nat-omega-simps*)
show *ArrVal-vdomain*:
$\mathcal{D}_{\circ}$ (*ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$) =
ntcf-pointed-inv α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrDom|\!)$
unfolding *app-X arr-Rel-components* **by** *simp*
from *assms* **show** *vrange-left*:
$\mathcal{R}_{\circ}$ (*ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrVal|\!)$) $\subseteq_{\circ}$
ntcf-pointed-inv α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrCod|\!)$
unfolding *app-X arr-Rel-components* **by** (*auto intro*: *cat-cs-intros*)
from *assms* **show** *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)(\!|ArrCod|\!)$ $\in_{\circ}$ *Vset* α
unfolding *app-X arr-Rel-components cat-Set-components*(*1*)
by (*intro Set.cat-Hom-in-Vset*[*OF - assms*(*2*)])
(*auto simp*: *cat-Set-components*(*1*))
qed (*use assms* **in** ‹*auto simp*: *app-X arr-Rel-components cat-Set-components*(*1*)›)
qed

lemma (**in** $\mathcal{Z}$) *ntcf-pointed-inv-NTMap-app-is-arr'*[*cat-cs-intros*]:
assumes $\mathfrak{a} \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
and $X \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
and $A' = X$
and $B' =$ *Hom* (*cat-Set* α) (*set* $\{\mathfrak{a}\}$) X
and $\mathfrak{C}' =$ *cat-Set* α
shows *ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!) : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1*,*2*)
unfolding *assms*(*3*–*5*)
by (*rule ntcf-pointed-inv-NTMap-app-is-arr*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$.*ntcf-pointed-inv-NTMap-app-is-arr'*

lemma (**in** $\mathcal{Z}$) *is-inverse-ntcf-pointed-inv-NTMap-app*:
assumes $\mathfrak{a} \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $X \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
shows
is-inverse
(*cat-Set* α)
(*ntcf-pointed-inv* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)$)
(*ntcf-pointed* α $\mathfrak{a}(\!|NTMap|\!)(\!|X|\!)$)
(**is** ‹*is-inverse* (*cat-Set* α) *?bwd ?fwd*›)
proof(*intro is-inverseI*)

let *?Hom* = ‹*Hom* (*cat-Set* α) (*set* $\{\mathfrak{a}\}$) X›

interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)

from *assms*(*1*) **have** *set-*$\mathfrak{a}$: *set* $\{\mathfrak{a}\} \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
unfolding *cat-Set-components*(*1*) **by** *blast*
have *Hom-*$\mathfrak{a}X$: *?Hom* $\in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
by
(
auto
simp: *cat-Set-components*(*1*)
intro!: *Set.cat-Hom-in-Vset set-*$\mathfrak{a}$ *assms*(*2*)
)

from *assms* **show** *?bwd* : $X \mapsto_{\textit{cat-Set}\ \alpha}$ *?Hom*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **show** *?fwd* : *?Hom* $\mapsto_{\textit{cat-Set}\ \alpha} X$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

from *assms set-*$\mathfrak{a}$ *Hom-*$\mathfrak{a}X$ **have** *lhs-is-arr*:
?bwd $\circ_{A\,\textit{cat-Set}\ \alpha}$ *?fwd* : *?Hom* $\mapsto_{\textit{cat-Set}\ \alpha}$ *?Hom*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_{\circ}$ ((*?bwd* $\circ_{A\,\textit{cat-Set}\ \alpha}$ *?fwd*)(|*ArrVal*|)) = *?Hom*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)

from *Hom-*$\mathfrak{a}X$ **have** *rhs-is-arr*: *cat-Set* $\alpha(\!|CId|\!)(\!|?Hom|\!)$: *?Hom* $\mapsto_{\textit{cat-Set}\ \alpha}$ *?Hom*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_{\circ}$ ((*cat-Set* $\alpha(\!|CId|\!)(\!|?Hom|\!)$)(|*ArrVal*|)) = *?Hom*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)

show *?bwd* $\circ_{A\,\textit{cat-Set}\ \alpha}$ *?fwd* = *cat-Set* $\alpha(\!|CId|\!)(\!|?Hom|\!)$
proof(*rule arr-Set-eqI*)
from *lhs-is-arr* **show** *arr-Set-lhs*: *arr-Set* α (*?bwd* $\circ_{A\,\textit{cat-Set}\ \alpha}$ *?fwd*)
by (*auto dest*: *cat-Set-is-arrD*)
from *rhs-is-arr* **show** *arr-Set-rhs*: *arr-Set* α (*cat-Set* $\alpha(\!|CId|\!)(\!|?Hom|\!)$)

```
    by (auto dest: cat-Set-is-arrD)
  show (?bwd ∘_A cat-Set α ?fwd)(|ArrVal|) = cat-Set α(|CId|)(|?Hom|)(|ArrVal|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
    fix F assume F : set {𝔞} ↦_cat-Set α X
    with assms Hom-𝔞X show
      (?bwd ∘_A cat-Set α ?fwd)(|ArrVal|)(|F|) = cat-Set α(|CId|)(|?Hom|)(|ArrVal|)(|F|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps ntcf-paa-ArrVal
            cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
        )
  qed (use arr-Set-lhs arr-Set-rhs in auto)
qed (use lhs-is-arr rhs-is-arr in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

from assms have lhs-is-arr: ?fwd ∘_A cat-Set α ?bwd : X ↦_cat-Set α X
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
then have dom-lhs: 𝒟∘ ((?fwd ∘_A cat-Set α ?bwd)(|ArrVal|)) = X
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)

from assms have rhs-is-arr: cat-Set α(|CId|)(|X|) : X ↦_cat-Set α X
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
then have dom-rhs: 𝒟∘ ((cat-Set α(|CId|)(|X|))(|ArrVal|)) = X
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)

show ?fwd ∘_A cat-Set α ?bwd = cat-Set α(|CId|)(|X|)
proof(rule arr-Set-eqI)
  from lhs-is-arr show arr-Set-lhs: arr-Set α (?fwd ∘_A cat-Set α ?bwd)
    by (auto dest: cat-Set-is-arrD)
  from rhs-is-arr show arr-Set-rhs: arr-Set α (cat-Set α(|CId|)(|X|))
    by (auto dest: cat-Set-is-arrD)
  show (?fwd ∘_A cat-Set α ?bwd)(|ArrVal|) = cat-Set α(|CId|)(|X|)(|ArrVal|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix x assume x ∈∘ X
    with assms show
      (?fwd ∘_A cat-Set α ?bwd)(|ArrVal|)(|x|) = cat-Set α(|CId|)(|X|)(|ArrVal|)(|x|)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed (use arr-Set-lhs arr-Set-rhs in auto)
qed (use lhs-is-arr rhs-is-arr in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed
```

8.3.3 Inverse pointed natural transformation is a natural isomorphism

```
lemma (in 𝒵) ntcf-pointed-inv-is-ntcf:
  assumes 𝔞 ∈∘ cat-Set α(|Obj|)
  shows ntcf-pointed-inv α 𝔞 :
    cf-id (cat-Set α) ↦_CF Hom_O.Cα cat-Set α(set {𝔞},−) :
    cat-Set α ↦↦_Cα cat-Set α
proof(intro is-ntcfI')

  interpret Set: category α ‹cat-Set α› by (rule category-cat-Set)

  note 𝔞 = assms[unfolded cat-Set-components(1)]
  from assms have set-𝔞: set {𝔞} ∈∘ cat-Set α(|Obj|)
    unfolding cat-Set-components by auto

  show vfsequence (ntcf-pointed-inv α 𝔞)
```

```
  unfolding ntcf-pointed-inv-def by simp
show vcard (ntcf-pointed-inv α 𝔞) = 5ₙ
  unfolding ntcf-pointed-inv-def by (simp add: nat-omega-simps)
from set-𝔞 show Hom_{O.Cα}cat-Set α(set {𝔞},−) : cat-Set α ↦↦_{Cα} cat-Set α
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)

show ntcf-pointed-inv α 𝔞(|NTMap|)(|a|) :
  cf-id (cat-Set α)(|ObjMap|)(|a|) ↦_{cat-Set α}
  Hom_{O.Cα}cat-Set α(set {𝔞},−)(|ObjMap|)(|a|)
  if a ∈₀ cat-Set α(|Obj|) for a
  using that assms set-𝔞
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
    )

show
  ntcf-pointed-inv α 𝔞(|NTMap|)(|B|) ∘_{A cat-Set α} cf-id (cat-Set α)(|ArrMap|)(|F|) =
    Hom_{O.Cα}cat-Set α(set {𝔞},−)(|ArrMap|)(|F|) ∘_{A cat-Set α}
      ntcf-pointed-inv α 𝔞(|NTMap|)(|A|)
  if F : A ↦_{cat-Set α} B for A B F
proof–
  let ?pb = ‹ntcf-pointed-inv α 𝔞(|NTMap|)(|B|)›
    and ?pa = ‹ntcf-pointed-inv α 𝔞(|NTMap|)(|A|)›
    and ?hom = ‹cf-hom (cat-Set α) [cat-Set α(|CId|)(|set {𝔞}|), F]₀›
  from assms set-𝔞 that have pb-F:
    ?pb ∘_{A cat-Set α} F : A ↦_{cat-Set α} Hom (cat-Set α) (set {𝔞}) B
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-prod-cs-intros)
  then have dom-lhs: 𝒟₀ ((?pb ∘_{A cat-Set α} F)(|ArrVal|)) = A
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from that assms set-𝔞 that have hom-pa:
    ?hom ∘_{A cat-Set α} ?pa : A ↦_{cat-Set α} Hom (cat-Set α) (set {𝔞}) B
    by (cs-concl cs-intro: cat-cs-intros cat-prod-cs-intros cat-op-intros)
  then have dom-rhs: 𝒟₀ ((?hom ∘_{A cat-Set α} ?pa)(|ArrVal|)) = A
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have [cat-cs-simps]: ?pb ∘_{A cat-Set α} F = ?hom ∘_{A cat-Set α} ?pa
  proof(rule arr-Set-eqI)
    from pb-F show arr-Set-pb-F: arr-Set α (?pb ∘_{A cat-Set α} F)
      by (auto dest: cat-Set-is-arrD(1))
    from hom-pa show arr-Set-hom-pa: arr-Set α (?hom ∘_{A cat-Set α} ?pa)
      by (auto dest: cat-Set-is-arrD(1))
    show (?pb ∘_{A cat-Set α} F)(|ArrVal|) = (?hom ∘_{A cat-Set α} ?pa)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈₀ A
      with assms 𝔞 set-𝔞 that show
        (?pb ∘_{A cat-Set α} F)(|ArrVal|)(|a|) = (?hom ∘_{A cat-Set α} ?pa)(|ArrVal|)(|a|)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps
              cs-intro:
                cat-Set-cs-intros
                cat-cs-intros
                cat-prod-cs-intros
                cat-op-intros
          )
    qed (use arr-Set-pb-F arr-Set-hom-pa in auto)
```

```
    qed (use pb-F hom-pa in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
    from assms that set-𝔞 show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
        )
  qed

qed (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

lemma (in 𝒵) ntcf-pointed-inv-is-ntcf'[cat-cs-intros]:
  assumes 𝔞 ∈₀ cat-Set α(|Obj|)
    and 𝔉' = cf-id (cat-Set α)
    and 𝔊' = Hom_{O.Cα}cat-Set α(set {𝔞},−)
    and 𝔄' = cat-Set α
    and 𝔅' = cat-Set α
    and α' = α
  shows ntcf-pointed-inv α 𝔞 : 𝔉' ↦_{CF} 𝔊' : 𝔄' ↦↦_{C_α'} 𝔅'
  using assms(1) unfolding assms(2–6) by (rule ntcf-pointed-inv-is-ntcf)

lemmas [cat-cs-intros] = 𝒵.ntcf-pointed-inv-is-ntcf'

lemma (in 𝒵) inv-ntcf-ntcf-pointed[cat-cs-simps]:
  assumes 𝔞 ∈₀ cat-Set α(|Obj|)
  shows inv-ntcf (ntcf-pointed α 𝔞) = ntcf-pointed-inv α 𝔞
  by
    (
      rule iso-ntcf-if-is-inverse(3)[symmetric],
      rule is-iso-ntcfD(1)[OF ntcf-pointed-is-iso-ntcf[OF assms]],
      rule ntcf-pointed-inv-is-ntcf[OF assms],
      rule is-inverse-ntcf-pointed-inv-NTMap-app[OF assms]
    )

lemma (in 𝒵) inv-ntcf-ntcf-pointed-inv[cat-cs-simps]:
  assumes 𝔞 ∈₀ cat-Set α(|Obj|)
  shows inv-ntcf (ntcf-pointed-inv α 𝔞) = ntcf-pointed α 𝔞
  by
    (
      rule iso-ntcf-if-is-inverse(4)[symmetric],
      rule is-iso-ntcfD(1)[OF ntcf-pointed-is-iso-ntcf[OF assms]],
      rule ntcf-pointed-inv-is-ntcf[OF assms],
      rule is-inverse-ntcf-pointed-inv-NTMap-app[OF assms]
    )

lemma (in 𝒵) ntcf-pointed-inv-is-iso-ntcf:
  assumes 𝔞 ∈₀ cat-Set α(|Obj|)
  shows ntcf-pointed-inv α 𝔞 :
    cf-id (cat-Set α) ↦_{CF.iso} Hom_{O.Cα}cat-Set α(set {𝔞},−) :
    cat-Set α ↦↦_{Cα} cat-Set α
  by
    (
      rule iso-ntcf-if-is-inverse(2),
      rule is-iso-ntcfD(1)[OF ntcf-pointed-is-iso-ntcf[OF assms]],
      rule ntcf-pointed-inv-is-ntcf[OF assms],
      rule is-inverse-ntcf-pointed-inv-NTMap-app[OF assms]
    )
```

lemma (**in** $\mathcal{Z}$) *ntcf-pointed-inv-is-iso-ntcf'*[*cat-cs-intros*]:
assumes $\mathfrak{a} \in_{\circ}$ *cat-Set* $\alpha(\lfloor Obj \rfloor)$
and $\mathfrak{F}' =$ *cf-id* (*cat-Set* α)
and $\mathfrak{G}' = Hom_{O.C\alpha}$*cat-Set* α(*set* $\{\mathfrak{a}\}$,−)
and $\mathfrak{A}' =$ *cat-Set* α
and $\mathfrak{B}' =$ *cat-Set* α
and $\alpha' = \alpha$
shows *ntcf-pointed-inv* α $\mathfrak{a} : \mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C_{\alpha'}} \mathfrak{B}'$
using *assms*(*1*) **unfolding** *assms*(*2–6*) **by** (*rule ntcf-pointed-inv-is-iso-ntcf*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$*.ntcf-pointed-inv-is-iso-ntcf'*

9 Representable and corepresentable functors

9.1 Representable and corepresentable functors

9.1.1 Definitions and elementary properties

See Chapter III-2 in [9] or Section 2.1 in [14].

definition *cat-representation* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *cat-representation* $\alpha\ \mathfrak{F}\ c\ \psi \longleftrightarrow$
    $c \in_\circ \mathfrak{F}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor) \wedge$
    $\psi : Hom_{O.C\alpha}\mathfrak{F}(\lfloor HomDom \rfloor)(c,-) \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{F}(\lfloor HomDom \rfloor) \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$

definition *cat-corepresentation* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *cat-corepresentation* $\alpha\ \mathfrak{F}\ c\ \psi \longleftrightarrow$
    $c \in_\circ \mathfrak{F}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor) \wedge$
    $\psi : Hom_{O.C\alpha} op\text{-}cat\ (\mathfrak{F}(\lfloor HomDom \rfloor))(-,c) \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{F}(\lfloor HomDom \rfloor) \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$

Rules.

context
  fixes $\alpha\ \mathfrak{C}\ \mathfrak{F}$
  assumes $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
begin

interpretation $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}\ \langle cat\text{-}Set\ \alpha\rangle\ \mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)

mk-ide rf *cat-representation-def*[**where** α=α **and** $\mathfrak{F}$=$\mathfrak{F}$, *unfolded cat-cs-simps*]
  |*intro cat-representationI*|
  |*dest cat-representationD'*|
  |*elim cat-representationE'*|

end

lemmas *cat-representationD*[*dest*] = *cat-representationD'*[*rotated*]
  and *cat-representationE*[*elim*] = *cat-representationE'*[*rotated*]

lemma *cat-corepresentationI*:
  assumes *category* $\alpha\ \mathfrak{C}$
    and $\mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
    and $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
    and $\psi : Hom_{O.C\alpha}\mathfrak{C}(-,c) \mapsto_{CF.iso} \mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
  shows *cat-corepresentation* $\alpha\ \mathfrak{F}\ c\ \psi$
proof–
  interpret *category* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{F}$: *is-functor* $\alpha\ \langle op\text{-}cat\ \mathfrak{C}\rangle\ \langle cat\text{-}Set\ \alpha\rangle\ \mathfrak{F}$ **by** (*rule assms*(*2*))
  note [*cat-op-simps*] = $\mathfrak{F}$.*HomDom.cat-op-cat-cf-Hom-snd*[
      symmetric, *unfolded cat-op-simps*, *OF assms*(*3*)
      ]
  show *?thesis*
    unfolding *cat-corepresentation-def*
    by (*intro conjI*, *unfold cat-cs-simps cat-op-simps*; *intro assms*)
qed

lemma *cat-corepresentationD*:
  assumes *cat-corepresentation* $\alpha\ \mathfrak{F}\ c\ \psi$
    and *category* $\alpha\ \mathfrak{C}$
    and $\mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
  shows $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
    and $\psi : Hom_{O.C\alpha}\mathfrak{C}(-,c) \mapsto_{CF.iso} \mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$

proof–
interpret *category* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-functor* α ‹*op-cat* $\mathfrak{C}$› ‹*cat-Set* α› $\mathfrak{F}$ **by** (*rule assms*(*3*))
note *cψ* = *cat-corepresentation-def*[
THEN iffD1, *OF assms*(*1*), *unfolded cat-cs-simps cat-op-simps*
]
from *cψ*(*1*) **show** *c*: $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
note [*cat-op-simps*] = $\mathfrak{F}$.*HomDom.cat-op-cat-cf-Hom-snd*[
symmetric, *unfolded cat-op-simps*, *OF c*
]
show $\psi : Hom_{O.C\alpha}\mathfrak{C}(-,c) \mapsto_{CF.iso} \mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*rule conjunct2*[*OF cψ*, *unfolded cat-op-simps*])
qed

lemma *cat-corepresentationE*:
assumes *cat-corepresentation* α $\mathfrak{F}$ *c* ψ
and *category* α $\mathfrak{C}$
and $\mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
obtains $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $\psi : Hom_{O.C\alpha}\mathfrak{C}(-,c) \mapsto_{CF.iso} \mathfrak{F} : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*simp add*: *cat-corepresentationD*[*OF assms*])

9.1.2 Representable functors and universal arrows

lemma *universal-arrow-of-if-cat-representation*:
— See Proposition 2 in Chapter III-2 in [9].
assumes $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
and *cat-representation* α $\mathfrak{K}$ *r* ψ
and $\mathfrak{a} \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$
shows *universal-arrow-of*
$\mathfrak{K}$ (*set* {$\mathfrak{a}$}) *r* (*ntcf-paa* $\mathfrak{a}$ ($\mathfrak{K}(\!|ObjMap|\!)(\!|r|\!)$) ($\psi(\!|NTMap|\!)(\!|r|\!)(\!|ArrVal|\!)(\!|\mathfrak{C}(\!|CId|\!)(\!|r|\!)|\!)$))
proof–
note *rψ* = *cat-representationD*[*OF assms*(*2*,*1*)]
interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{K}$ **by** (*rule assms*(*1*))
interpret ψ: *is-iso-ntcf* α $\mathfrak{C}$ ‹*cat-Set* α› ‹$Hom_{O.C\alpha}\mathfrak{C}(r,-)$› $\mathfrak{K}$ ψ
by (*rule rψ*(*2*))
from *assms*(*3*) **have** *set-*$\mathfrak{a}$: *set* {$\mathfrak{a}$} $\in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
by (*simp add*: *Limit-vsingleton-in-VsetI cat-Set-components*(*1*))
from
ntcf-cf-comp-is-iso-ntcf[
OF $\mathfrak{K}$.*ntcf-pointed-inv-is-iso-ntcf*[*OF assms*(*3*)] *assms*(*1*)
]
have $\mathfrak{a}\mathfrak{K}$: *ntcf-pointed-inv* α $\mathfrak{a}$ $\circ_{NTCF-CF}$ $\mathfrak{K}$:
$\mathfrak{K} \mapsto_{CF.iso} Hom_{O.C\alpha}cat\text{-}Set\ \alpha$ (*set* {$\mathfrak{a}$},−) $\circ_{CF}$ $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*cs-prems* **cs-simp**: *cat-cs-simps*)
from *iso-ntcf-is-iso-arr*(*1*)[*OF* $\mathfrak{a}\mathfrak{K}$] *rψ assms*(*3*) **have** [*cat-cs-simps*]:
((*ntcf-pointed-inv* α $\mathfrak{a}$ $\circ_{NTCF-CF}$ $\mathfrak{K}$ $\bullet_{NTCF}$ ψ)$(\!|NTMap|\!)(\!|r|\!)(\!|ArrVal|\!)(\!|\mathfrak{C}(\!|CId|\!)(\!|r|\!)|\!)$ =
ntcf-paa $\mathfrak{a}$ ($\mathfrak{K}(\!|ObjMap|\!)(\!|r|\!)$) ($\psi(\!|NTMap|\!)(\!|r|\!)(\!|ArrVal|\!)(\!|\mathfrak{C}(\!|CId|\!)(\!|r|\!)|\!)$)
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-Set-cs-intros cat-cs-intros cat-op-intros*
)
show *universal-arrow-of*
$\mathfrak{K}$ (*set* {$\mathfrak{a}$}) *r* (*ntcf-paa* $\mathfrak{a}$ ($\mathfrak{K}(\!|ObjMap|\!)(\!|r|\!)$) ($\psi(\!|NTMap|\!)(\!|r|\!)(\!|ArrVal|\!)(\!|\mathfrak{C}(\!|CId|\!)(\!|r|\!)|\!)$))
by
(

```
    rule 𝔎.cf-universal-arrow-of-if-is-iso-ntcf
      [
        OF rψ(1) set-𝔞 ntcf-vcomp-is-iso-ntcf[OF 𝔞𝔎 rψ(2)],
        unfolded cat-cs-simps
      ]
  )
qed

lemma universal-arrow-of-if-cat-corepresentation:
  — See Proposition 2 in Chapter III-2 in [9].
  assumes category α ℭ
    and 𝔎 : op-cat ℭ ↦↦_Cα cat-Set α
    and cat-corepresentation α 𝔎 r ψ
    and 𝔞 ∈₀ cat-Set α(|Obj|)
  shows universal-arrow-of
    𝔎 (set {𝔞}) r (ntcf-paa 𝔞 (𝔎(|ObjMap|)(|r|)) (ψ(|NTMap|)(|r|)(|ArrVal|)(|ℭ(|CId|)(|r|))))
proof–
  interpret ℭ: category α ℭ by (rule assms(1))
  note rψ = cat-corepresentationD[OF assms(3,1,2)]
  note [cat-op-simps] = ℭ.cat-op-cat-cf-Hom-snd[OF rψ(1)]
  have rep: cat-representation α 𝔎 r ψ
    by (intro cat-representationI, rule assms(2), unfold cat-op-simps; rule rψ)
  show ?thesis
    by
      (
        rule universal-arrow-of-if-cat-representation[
          OF assms(2) rep assms(4), unfolded cat-op-simps
        ]
      )
qed

lemma cat-representation-if-universal-arrow-of:
  — See Proposition 2 in Chapter III-2 in [9].
  assumes 𝔎 : ℭ ↦↦_Cα cat-Set α
    and 𝔞 ∈₀ cat-Set α(|Obj|)
    and universal-arrow-of 𝔎 (set {𝔞}) r u
  shows cat-representation α 𝔎 r (Yoneda-arrow α 𝔎 r (u(|ArrVal|)(|𝔞|)))
proof–

  let ?Y = ‹Yoneda-component 𝔎 r (u(|ArrVal|)(|𝔞|))›

  interpret 𝔎: is-functor α ℭ ‹cat-Set α› 𝔎 by (rule assms(1))

  note ua = 𝔎.universal-arrow-ofD[OF assms(3)]

  from ua(2) have u𝔞: u(|ArrVal|)(|𝔞|) ∈₀ 𝔎(|ObjMap|)(|r|)
    by
      (
        cs-concl cs-shallow
          cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
      )

  have [cat-cs-simps]: Yoneda-arrow α 𝔎 r (u(|ArrVal|)(|𝔞|))(|NTMap|)(|c|) = ?Y c
    if c ∈₀ ℭ(|Obj|) for c
    using that
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from ua(1) have [cat-cs-simps]: Hom_O.Cα ℭ(r,–)(|ObjMap|)(|c|) = Hom ℭ r c
    if c ∈₀ ℭ(|Obj|) for c
```

```
  using that
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-op-intros)

show ?thesis
proof
  (
    intro cat-representationI is-iso-ntcfI,
    rule assms(1),
    rule ua(1),
    rule ℜ.HomDom.cat-Yoneda-arrow-is-ntcf[OF assms(1) ua(1) u𝔞],
    rule cat-Set-is-iso-arrI,
    simp-all only: cat-cs-simps
  )

  fix c assume prems: c ∈₀ ℭ⦇Obj⦈
  with ua(1,2) show Yc: ?Y c : Hom ℭ r c ↦_cat-Set α ℜ⦇ObjMap⦈⦇c⦈
    by
      (
        cs-concl cs-shallow
          cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
      )

  note YcD = cat-Set-is-arrD[OF Yc]

  interpret Yc: arr-Set α ‹?Y c› by (rule YcD(1))

  show dom-Yc: 𝒟₀ (?Y c⦇ArrVal⦈) = Hom ℭ r c
    by (simp add: ℜ.Yoneda-component-ArrVal-vdomain)

  show v11 (?Y c⦇ArrVal⦈)
  proof(intro Yc.ArrVal.vsv-valeq-v11I, unfold dom-Yc in-Hom-iff)

    fix g f assume prems':
      g : r ↦_ℭ c f : r ↦_ℭ c ?Y c⦇ArrVal⦈⦇g⦈ = ?Y c⦇ArrVal⦈⦇f⦈

    from prems have c: c ∈₀ ℭ⦇Obj⦈ by auto

    from prems'(3,1,2) have ℜgu𝔞-ℜfu𝔞:
      ℜ⦇ArrMap⦈⦇g⦈⦇ArrVal⦈⦇u⦇ArrVal⦈⦇𝔞⦈⦈ = ℜ⦇ArrMap⦈⦇f⦈⦇ArrVal⦈⦇u⦇ArrVal⦈⦇𝔞⦈⦈
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

    from prems'(1,2) ua(1,2) have ℜg-u:
      ℜ⦇ArrMap⦈⦇g⦈ ∘_A cat-Set α u : set {𝔞} ↦_cat-Set α ℜ⦇ObjMap⦈⦇c⦈
      and ℜf-u:
      ℜ⦇ArrMap⦈⦇f⦈ ∘_A cat-Set α u : set {𝔞} ↦_cat-Set α ℜ⦇ObjMap⦈⦇c⦈
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
    then have dom-lhs: 𝒟₀ ((ℜ⦇ArrMap⦈⦇g⦈ ∘_A cat-Set α u)⦇ArrVal⦈) = set {𝔞}
      and dom-rhs: 𝒟₀ ((ℜ⦇ArrMap⦈⦇f⦈ ∘_A cat-Set α u)⦇ArrVal⦈) = set {𝔞}
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)+

    have ℜg-ℜf: ℜ⦇ArrMap⦈⦇g⦈ ∘_A cat-Set α u = ℜ⦇ArrMap⦈⦇f⦈ ∘_A cat-Set α u
    proof(rule arr-Set-eqI)
      from ℜg-u show arr-Set-ℜg-u: arr-Set α (ℜ⦇ArrMap⦈⦇g⦈ ∘_A cat-Set α u)
        by (auto dest: cat-Set-is-arrD)
      from ℜf-u show arr-Set-ℜf-u: arr-Set α (ℜ⦇ArrMap⦈⦇f⦈ ∘_A cat-Set α u)
        by (auto dest: cat-Set-is-arrD)
      show
        (ℜ⦇ArrMap⦈⦇g⦈ ∘_A cat-Set α u)⦇ArrVal⦈ =
```

```
      (ℜ(|ArrMap|)(|f|) ∘_A cat-Set α u)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs vsingleton-iff; (simp only:) ?)
      from prems'(1,2) ua(2) show
        (ℜ(|ArrMap|)(|g|) ∘_A cat-Set α u)(|ArrVal|)(|𝔞|) =
          (ℜ(|ArrMap|)(|f|) ∘_A cat-Set α u)(|ArrVal|)(|𝔞|)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps ℜgu𝔞-ℜfu𝔞
              cs-intro: V-cs-intros cat-cs-intros
          )
    qed (use arr-Set-ℜg-u arr-Set-ℜf-u in auto)
  qed (use ℜg-u ℜf-u in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
  from prems'(1) ua(2) have
    ℜ(|ArrMap|)(|g|) ∘_A cat-Set α u : set {𝔞} ↦_cat-Set α ℜ(|ObjMap|)(|c|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from ua(3)[OF c this] obtain h where h: h : r ↦_ℭ c
    and ℜg-u-def:
      ℜ(|ArrMap|)(|g|) ∘_A cat-Set α u = umap-of ℜ (set {𝔞}) r u c(|ArrVal|)(|h|)
    and h-unique: ⋀h'.
      [[
        h' : r ↦_ℭ c;
        ℜ(|ArrMap|)(|g|) ∘_A cat-Set α u = umap-of ℜ (set {𝔞}) r u c(|ArrVal|)(|h'|)
      ]] ⟹ h' = h
    by metis
  from prems'(1,2) ua(2) have
    ℜ(|ArrMap|)(|g|) ∘_A cat-Set α u = umap-of ℜ (set {𝔞}) r u c(|ArrVal|)(|g|)
    ℜ(|ArrMap|)(|g|) ∘_A cat-Set α u = umap-of ℜ (set {𝔞}) r u c(|ArrVal|)(|f|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps ℜg-ℜf cs-intro: cat-cs-intros
      )+
  from h-unique[OF prems'(1) this(1)] h-unique[OF prems'(2) this(2)] show
    g = f
    by simp
qed

show ℛ∘ (?Y c(|ArrVal|)) = ℜ(|ObjMap|)(|c|)
proof
  (
    intro
      vsubset-antisym Yc.arr-Par-ArrVal-vrange[unfolded YcD]
      vsubsetI
  )
  fix y assume prems': y ∈∘ ℜ(|ObjMap|)(|c|)
  from prems have ℜc: ℜ(|ObjMap|)(|c|) ∈∘ cat-Set α(|Obj|)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from ua(3)[OF prems ℜ.ntcf-paa-is-arr[OF assms(2) ℜc prems']] obtain f
    where f: f : r ↦_ℭ c
    and ntcf-paa-y:
      ntcf-paa 𝔞 (ℜ(|ObjMap|)(|c|)) y = umap-of ℜ (set {𝔞}) r u c(|ArrVal|)(|f|)
    and f-unique: ⋀f'.
      [[
        f' : r ↦_ℭ c;
        ntcf-paa 𝔞 (ℜ(|ObjMap|)(|c|)) y = umap-of ℜ (set {𝔞}) r u c(|ArrVal|)(|f'|)
      ]] ⟹ f' = f
    by metis
```

```
    from ntcf-paa-y f ua(2) have
      ntcf-paa 𝔞 (ℜ(|ObjMap|)(|c|)) y = ℜ(|ArrMap|)(|f|) ∘_A cat-Set α u
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    then have
      ntcf-paa 𝔞 (ℜ(|ObjMap|)(|c|)) y(|ArrVal|)(|𝔞|) =
        (ℜ(|ArrMap|)(|f|) ∘_A cat-Set α u)(|ArrVal|)(|𝔞|)
      by simp
    from this f ua(2) have [symmetric, cat-cs-simps]:
      y = ℜ(|ArrMap|)(|f|)(|ArrVal|)(|u(|ArrVal|)(|𝔞|)|)
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
        )
    show y ∈_∘ ℛ_∘ (?Y c(|ArrVal|))
      by (intro Yc.ArrVal.vsv-vimageI2')
        (
          use f in
            ‹
              cs-concl cs-shallow
                cs-simp: cat-cs-simps cs-intro: cat-cs-intros
            ›
        )+
  qed
qed

qed

lemma cat-corepresentation-if-universal-arrow-of:
  — See Proposition 2 in Chapter III-2 in [9].
  assumes category α 𝔈
    and ℜ : op-cat 𝔈 ↦↦_Cα cat-Set α
    and 𝔞 ∈_∘ cat-Set α(|Obj|)
    and universal-arrow-of ℜ (set {𝔞}) r u
  shows cat-corepresentation α ℜ r (Yoneda-arrow α ℜ r (u(|ArrVal|)(|𝔞|)))
proof–
  interpret 𝔈: category α 𝔈 by (rule assms(1))
  interpret ℜ: is-functor α ‹op-cat 𝔈› ‹cat-Set α› ℜ by (rule assms(2))
  note ua = ℜ.universal-arrow-ofD[OF assms(4), unfolded cat-op-simps]
  note [cat-op-simps] = 𝔈.cat-op-cat-cf-Hom-snd[OF ua(1)]
  show ?thesis
    by
      (
        intro cat-corepresentationI,
        rule assms(1),
        rule assms(2),
        rule ua(1),
        rule cat-representationD(2)
          [
            OF
              cat-representation-if-universal-arrow-of[OF assms(2,3,4)]
              assms(2),
            unfolded cat-op-simps
          ]
      )
qed
```

9.2 Limits and colimits as universal cones

lemma *is-tm-cat-limit-if-cat-corepresentation*:
— See Definition 3.1.5 in Section 3.1 in [14].
assumes $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and *cat-corepresentation* α (*tm-cf-Cone* α $\mathfrak{F}$) r ψ
(**is** ‹*cat-corepresentation* α *?Cone* r ψ›)
shows *ntcf-of-ntcf-arrow* $\mathfrak{J}$ $\mathfrak{C}$ (ψ(|*NTMap*|)(|r|)(|*ArrVal*|)(|$\mathfrak{C}$(|*CId*|)(|r|)|)) :
r <$_{CF.tm.lim}$ $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
(**is** ‹*ntcf-of-ntcf-arrow* $\mathfrak{J}$ $\mathfrak{C}$ *?ψr1r* : r <$_{CF.tm.lim}$ $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$›)
proof–

let *?P* = ‹*ntcf-paa 0*› **and** *?Funct* = ‹*cat-Funct* α $\mathfrak{J}$ $\mathfrak{C}$›

interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret *Funct*: *category* α *?Funct*
by
(
cs-concl **cs-shallow cs-intro**:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)
note $r\psi$ = *cat-corepresentationD*[
OF assms(*2*) $\mathfrak{F}$.*HomCod.category-axioms* $\mathfrak{F}$.*tm-cf-cf-Cone-is-functor*
]
interpret ψ: *is-iso-ntcf* α ‹*op-cat* $\mathfrak{C}$› ‹*cat-Set* α› ‹$Hom_{O.C\alpha}\mathfrak{C}(-,r)$› *?Cone* ψ
by (*rule* $r\psi$(*2*))
have *0*: *0* $\in_\circ$ *cat-Set* α(|*Obj*|) **unfolding** *cat-Set-components* **by** *auto*
note *ua* = *universal-arrow-of-if-cat-corepresentation*[
OF $\mathfrak{F}$.*HomCod.category-axioms* $\mathfrak{F}$.*tm-cf-cf-Cone-is-functor assms*(*2*) *0*
]

show *?thesis*
proof(*rule is-tm-cat-limitI'*)

from $r\psi$(*1*) **have** [*cat-FUNCT-cs-simps*]:
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r)) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r
by
(
cs-concl
cs-simp: *cat-FUNCT-cs-simps*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)
from ψ.*ntcf-NTMap-is-arr*[*unfolded cat-op-simps*, *OF* $r\psi$(*1*)] $r\psi$(*1*) **have**
ψ(|*NTMap*|)(|r|) :
Hom $\mathfrak{C}$ r r $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom ?Funct* (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r)) (*cf-map* $\mathfrak{F}$)
by
(
cs-prems
cs-simp: *cat-small-cs-simps cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros*
)
with $r\psi$(*1*) **have** ψr-r:
?ψr1r : *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r) $\mapsto_{?Funct}$ *cf-map* $\mathfrak{F}$
by
(
cs-concl **cs-shallow cs-intro**:
cat-Set-cs-intros cat-cs-intros in-Hom-iff[*symmetric*]
)

from *rψ(1) cat-Funct-is-arrD(1)*[*OF ψr-r, unfolded cat-FUNCT-cs-simps*]
show *ntcf-of-ntcf-arrow* $\mathfrak{J}$ $\mathfrak{C}$ *?ψr1r* : $r <_{CF.tm.cone}$ $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro is-tm-cat-coneI*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-small-cs-intros*)

fix r' u' **assume** u' : $r' <_{CF.tm.cone}$ $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
then interpret u': *is-tm-cat-cone* α r' $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ u' **.**

have *Cone-r'*: *tm-cf-Cone* α $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r' \rfloor) \in_\circ$ *cat-Set* $\alpha(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-intro**: *cat-lim-cs-intros cat-cs-intros cat-op-intros*)
from *rψ(1)* **have** *Cone-r*: *tm-cf-Cone* α $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor) \in_\circ$ *cat-Set* $\alpha(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)
from *rψ(1)* **have** *ψr1r*:
$\psi(\lfloor NTMap \rfloor)(\lfloor r \rfloor)(\lfloor ArrVal \rfloor)(\lfloor \mathfrak{C}(\lfloor CId \rfloor)(\lfloor r \rfloor) \rfloor) \in_\circ$ *tm-cf-Cone* α $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-small-cs-simps cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros*
)
have *u'*: *ntcf-arrow u'* $\in_\circ$ *?Cone*$(\lfloor ObjMap \rfloor)(\lfloor r' \rfloor)$
by
(
cs-concl
cs-simp: *cat-small-cs-simps*
cs-intro: *cat-small-cs-intros cat-FUNCT-cs-intros cat-cs-intros*
)

have [*cat-cs-simps*]:
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* $\mathfrak{F}$) = $\mathfrak{F}$
cf-of-cf-map $\mathfrak{J}$ $\mathfrak{C}$ (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r)) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r
by (*cs-concl* **cs-simp**: *cat-FUNCT-cs-simps*)+

from *Cone-r 0 ψr1r rψ(1)* **have** *ψr1r-is-arr*: $\psi(\lfloor NTMap \rfloor)(\lfloor r \rfloor)(\lfloor ArrVal \rfloor)(\lfloor \mathfrak{C}(\lfloor CId \rfloor)(\lfloor r \rfloor) \rfloor)$:
cf-map (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r) $\mapsto_{?Funct}$ *cf-map* $\mathfrak{F}$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-small-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros*
)

from *rψ(1)* **have** [*cat-cs-intros*]:
Hom ?Funct (*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r)) (*cf-map* $\mathfrak{F}$) $\in_\circ$ *cat-Set* $\alpha(\lfloor Obj \rfloor)$
unfolding *cat-Set-components(1)*
by (*intro Funct.cat-Hom-in-Vset*)
(
cs-concl
cs-simp: *cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-FUNCT-cs-intros cat-cs-intros*
)+

note *ψr1r-is-arrD* = *cat-Funct-is-arrD*[*OF ψr1r-is-arr, unfolded cat-cs-simps*]

from *is-functor.universal-arrow-ofD(3)*
[
OF $\mathfrak{F}$*.tm-cf-cf-Cone-is-functor ua*,

```
      unfolded cat-op-simps,
      OF u'.cat-cone-obj ℱ.ntcf-paa-is-arr[OF 0 Cone-r' u']
    ]
  obtain f where f: f : r' ↦_ℭ r
    and Pf: ?P (?Cone(|ObjMap|)(|r'|)) (ntcf-arrow u') =
      umap-of ?Cone (set {0}) r (?P (?Cone(|ObjMap|)(|r|)) ?ψr1r) r'(|ArrVal|)(|f|)
    and f-unique: ⋀f'.
      [[
        f' : r' ↦_ℭ r;
        ?P (?Cone(|ObjMap|)(|r'|)) (ntcf-arrow u') =
          umap-of ?Cone (set {0}) r (?P (?Cone(|ObjMap|)(|r|)) ?ψr1r) r'(|ArrVal|)(|f'|)
      ]] ⟹ f' = f
    by metis

  show ∃!f.
    f : r' ↦_ℭ r ∧
    u' = ntcf-of-ntcf-arrow 𝔍 ℭ ?ψr1r ∘_NTCF ntcf-const 𝔍 ℭ f
  proof(intro ex1I conjI; (elim conjE)?)
    show f : r' ↦_ℭ r by (rule f)
    from Pf Cone-r 0 f ψr1r ψr1r-is-arr ψr1r-is-arrD(1) show
      u' = ntcf-of-ntcf-arrow 𝔍 ℭ ?ψr1r ∘_NTCF ntcf-const 𝔍 ℭ f
      by (subst (asm) ψr1r-is-arrD(2))
        (
          cs-prems
            cs-simp: cat-FUNCT-cs-simps cat-small-cs-simps cat-cs-simps
            cs-intro:
              cat-small-cs-intros
              cat-cs-intros
              cat-FUNCT-cs-intros
              cat-prod-cs-intros
              cat-op-intros
        )

    fix f' assume prems:
      f' : r' ↦_ℭ r
      u' = ntcf-of-ntcf-arrow 𝔍 ℭ ?ψr1r ∘_NTCF ntcf-const 𝔍 ℭ f'
    from Pf Cone-r 0 f ψr1r ψr1r-is-arr ψr1r-is-arrD(1) prems(1) have
      ?P (?Cone(|ObjMap|)(|r'|)) (ntcf-arrow u') =
        umap-of ?Cone (set {0}) r (?P (?Cone(|ObjMap|)(|r|)) ?ψr1r) r'(|ArrVal|)(|f'|)
      by (subst ψr1r-is-arrD(2))
        (
          cs-concl
            cs-simp: cat-FUNCT-cs-simps cat-small-cs-simps cat-cs-simps prems(2)
            cs-intro:
              cat-small-cs-intros
              cat-FUNCT-cs-intros
              cat-cs-intros
              cat-prod-cs-intros
              cat-op-intros
        )
    from f-unique[OF prems(1) this] show f' = f .
  qed

 qed

qed

lemma cat-corepresentation-if-is-tm-cat-limit:
```

— See Definition 3.1.5 in Section 3.1 in [14].
assumes $\psi : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
shows *cat-corepresentation*
α (*tm-cf-Cone* α $\mathfrak{F}$) r (*Yoneda-arrow* α (*tm-cf-Cone* α $\mathfrak{F}$) r (*ntcf-arrow* ψ))
(**is** ‹*cat-corepresentation* α *?Cone* r *?Y*ψ›)
proof–

let *?Funct* = ‹*cat-Funct* α $\mathfrak{J}$ $\mathfrak{C}$›
and *?P*-ψ = ‹*ntcf-paa* 0 (*?Cone*(|*ObjMap*|)(|r|)) (*ntcf-arrow* ψ)›
and *?ntcf-of* = ‹*ntcf-of-ntcf-arrow* $\mathfrak{J}$ $\mathfrak{C}$›

interpret ψ: *is-tm-cat-limit* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ r ψ **by** (*rule assms*(*1*))
interpret *Funct*: *category* α *?Funct*
by
(
cs-concl **cs-shallow cs-intro**:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)
interpret *Cone*: *is-functor* α ‹*op-cat* $\mathfrak{C}$› ‹*cat-Set* α› ‹*?Cone*›
by (*rule* ψ.*NTCod.tm-cf-cf-Cone-is-functor*)

have *0*: *0* $\in_\circ$ *cat-Set* α(|*Obj*|) **unfolding** *cat-Set-components* **by** *auto*
have *ntcf-arrow-*ψ:
ntcf-arrow ψ : *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r) $\mapsto_{?Funct}$ *cf-map* $\mathfrak{F}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-FUNCT-cs-intros*)
from ψ.*cat-cone-obj* *0 ntcf-arrow-*ψ **have** *P-*ψ:
?P-ψ : *set* {*0*} $\mapsto_{cat\text{-}Set\ \alpha}$ *?Cone*(|*ObjMap*|)(|r|)
by
(
cs-concl **cs-shallow**
cs-intro: *cat-cs-intros cat-op-intros*
cs-simp: *cat-small-cs-simps cat-FUNCT-cs-simps*
)

have *universal-arrow-of ?Cone* (*set* {*0*}) r *?P-*ψ
proof(*rule Cone.universal-arrow-ofI*, *unfold cat-op-simps*, *rule* ψ.*cat-cone-obj*)

from *0* ψ.*cat-cone-obj* **show** *?P-*ψ : *set* {*0*} $\mapsto_{cat\text{-}Set\ \alpha}$ *?Cone*(|*ObjMap*|)(|r|)
by
(
cs-concl
cs-intro:
cat-small-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
cs-simp: *cat-small-cs-simps*
)

fix r' u' **assume** *prems*:
$r' \in_\circ \mathfrak{C}$(|*Obj*|) u' : *set* {*0*} $\mapsto_{cat\text{-}Set\ \alpha}$ *?Cone*(|*ObjMap*|)(|r'|)

let *?const-r'* = ‹*cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ r')›
let *?Hom-r*$\mathfrak{F}$ = ‹*Hom ?Funct ?const-r'* (*cf-map* $\mathfrak{F}$)›

from *prems*(*2,1*) **have** *u'*: u' : *set* {*0*} $\mapsto_{cat\text{-}Set\ \alpha}$ *?Hom-r*$\mathfrak{F}$
by
(

```
    cs-prems cs-shallow
      cs-simp: cat-small-cs-simps cat-cs-simps cs-intro: cat-cs-intros
  )
from prems(1) have [cat-FUNCT-cs-simps]:
  cf-of-cf-map ℑ ℭ ?const-r′ = cf-const ℑ ℭ r′
  by
  (
    cs-concl
      cs-simp: cat-cs-simps cat-FUNCT-cs-simps cs-intro: cat-cs-intros
  )

from
  cat-Set-ArrVal-app-vrange[OF prems(2) vintersection-vsingleton]
  prems(1)
have u′(|ArrVal|)(|0|) : ?const-r′ ↦_?Funct cf-map 𝔉
  by (cs-prems cs-shallow cs-simp: cat-small-cs-simps cat-cs-simps)
note u′0 = cat-Funct-is-arrD[OF this, unfolded cat-FUNCT-cs-simps]

interpret u′0: is-tm-cat-cone α r′ ℑ ℭ 𝔉 ‹?ntcf-of (u′(|ArrVal|)(|0|))›
  by
  (
    rule is-tm-cat-coneI[
      OF is-tm-ntcfD(1)[OF u′0(1)] ψ.NTCod.is-tm-functor-axioms prems(1)
      ]
  )

from ψ.tm-cat-lim-ua-fo[OF u′0.is-cat-cone-axioms] obtain f
  where f: f : r′ ↦_ℭ r
    and u′0-def: ?ntcf-of (u′(|ArrVal|)(|0|)) = ψ ∘_NTCF ntcf-const ℑ ℭ f
    and f-unique: ⋀f′.
      [[
        f′ : r′ ↦_ℭ r;
        ?ntcf-of (u′(|ArrVal|)(|0|)) = ψ ∘_NTCF ntcf-const ℑ ℭ f′
      ]] ⟹ f′ = f
  by metis

note [cat-FUNCT-cs-simps] =
  ψ.ntcf-paa-ArrVal u′0(2)[symmetric] u′0-def[symmetric]

show ∃!f′.
  f′ : r′ ↦_ℭ r ∧ u′ = umap-of ?Cone (set {0}) r ?P-ψ r′(|ArrVal|)(|f′|)
proof(intro ex1I conjI; (elim conjE)?; (rule f)?)

  from f 0 u′ ntcf-arrow-ψ show
    u′ = umap-of ?Cone (set {0}) r ?P-ψ r′(|ArrVal|)(|f|)
    by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro:
          cat-small-cs-intros
          cat-FUNCT-cs-intros
          cat-prod-cs-intros
          cat-cs-intros
          cat-op-intros
        cs-simp: cat-FUNCT-cs-simps cat-small-cs-simps
    )
```

```
    fix f' assume prems':
      f' : r' ↦_ℭ r
      u' = umap-of ?Cone (set {0}) r ?P-ψ r'(|ArrVal|)(|f'|)

    let ?f' = ‹ntcf-const ℑ ℭ f'›

    from prems'(2,1) 0 ntcf-arrow-ψ P-ψ have
      u' = ntcf-paa 0 ?Hom-r𝔉 (ntcf-arrow (ψ ∙_NTCF ?f'))
      unfolding
        Cone.umap-of-ArrVal-app[unfolded cat-op-simps, OF prems'(1) P-ψ]
      by
        (
          cs-prems
            cs-simp: cat-FUNCT-cs-simps cat-small-cs-simps cat-cs-simps
            cs-intro:
              cat-small-cs-intros
              cat-FUNCT-cs-intros
              cat-prod-cs-intros
              cat-cs-intros
              cat-op-intros
        )
    then have
      ?ntcf-of (u'(|ArrVal|)(|0|)) =
        ?ntcf-of ((ntcf-paa 0 ?Hom-r𝔉 (ntcf-arrow (ψ ∙_NTCF ?f')))(|ArrVal|)(|0|))
      by simp
    from this prems'(1) have ?ntcf-of (u'(|ArrVal|)(|0|)) = ψ ∙_NTCF ?f'
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps cs-intro: cat-cs-intros
        )
    from f-unique[OF prems'(1) this] show f' = f .

  qed

qed

from
  cat-corepresentation-if-universal-arrow-of[
    OF ψ.NTDom.HomCod.category-axioms Cone.is-functor-axioms 0 this
    ]
show cat-corepresentation α ?Cone r ?Yψ
  by (cs-prems cs-shallow cs-simp: cat-cs-simps)

qed
```

10 Completeness and cocompleteness

10.1 Limits by products and equalizers

lemma *cat-limit-of-cat-prod-obj-and-cat-equalizer*:
— See Theorem 1 in Chapter V-2 in [9].
assumes $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and $\bigwedge \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}.\ [\![\ \mathfrak{f} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b};\ \mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}\ ]\!] \Longrightarrow$
$\exists E\ \varepsilon.\ \varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge A.$ *tm-cf-discrete* α $(\mathfrak{J}(\!|Obj|\!))$ A $\mathfrak{C} \Longrightarrow$
$\exists P\ \pi.\ \pi : P <_{CF.\prod} A : \mathfrak{J}(\!|Obj|\!) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge A.$ *tm-cf-discrete* α $(\mathfrak{J}(\!|Arr|\!))$ A $\mathfrak{C} \Longrightarrow$
$\exists P\ \pi.\ \pi : P <_{CF.\prod} A : \mathfrak{J}(\!|Arr|\!) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains $r\ u$ **where** $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

let *?L* $=\langle\lambda u.\ \mathfrak{F}(\!|ObjMap|\!)(\!|\mathfrak{J}(\!|Cod|\!)(\!|u|\!)|\!)\rangle$ **and** *?R* $=\langle\lambda i.\ \mathfrak{F}(\!|ObjMap|\!)(\!|i|\!)\rangle$

interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))

have *?R* $j \in_\circ \mathfrak{C}(\!|Obj|\!)$ **if** $j \in_\circ \mathfrak{J}(\!|Obj|\!)$ **for** j
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros that*)

have *tm-cf-discrete* α $(\mathfrak{J}(\!|Obj|\!))$ *?R* $\mathfrak{C}$
proof(*intro tm-cf-discreteI*)
show $\mathfrak{F}(\!|ObjMap|\!)(\!|i|\!) \in_\circ \mathfrak{C}(\!|Obj|\!)$ **if** $i \in_\circ \mathfrak{J}(\!|Obj|\!)$ **for** i
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros that*)
show *VLambda* $(\mathfrak{J}(\!|Obj|\!))$ *?R* $\in_\circ$ *Vset* α
proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
show $\mathcal{R}_\circ$ (*VLambda* $(\mathfrak{J}(\!|Obj|\!))$ *?R*) $\in_\circ$ *Vset* α
proof–
have $\mathcal{R}_\circ$ (*VLambda* $(\mathfrak{J}(\!|Obj|\!))$ *?R*) $\subseteq_\circ \mathcal{R}_\circ$ $(\mathfrak{F}(\!|ObjMap|\!))$
by (*auto simp*: $\mathfrak{F}$.*cf-ObjMap-vdomain*)
moreover have $\mathcal{R}_\circ$ $(\mathfrak{F}(\!|ObjMap|\!)) \in_\circ$ *Vset* α
by (*force intro*: *vrange-in-VsetI* $\mathfrak{F}$.*tm-cf-ObjMap-in-Vset*)
ultimately show *?thesis* **by** *auto*
qed
qed (*auto simp*: *cat-small-cs-intros*)
show $(\lambda i \in_\circ \mathfrak{J}(\!|Obj|\!).\ \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|i|\!)|\!)) \in_\circ$ *Vset* α
proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
show $\mathcal{R}_\circ$ $(\lambda i \in_\circ \mathfrak{J}(\!|Obj|\!).\ \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|i|\!)|\!)) \in_\circ$ *Vset* α
proof–
have $\mathcal{R}_\circ$ $(\lambda i \in_\circ \mathfrak{J}(\!|Obj|\!).\ \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|i|\!)|\!)) \subseteq_\circ \mathcal{R}_\circ$ $(\mathfrak{F}(\!|ArrMap|\!))$
proof(*rule vrange-VLambda-vsubset*)
fix x **assume** x: $x \in_\circ \mathfrak{J}(\!|Obj|\!)$
then have $\mathfrak{J}(\!|CId|\!)(\!|x|\!) \in_\circ \mathcal{D}_\circ$ $(\mathfrak{F}(\!|ArrMap|\!))$
by (*auto intro*: *cat-cs-intros simp*: *cat-cs-simps*)
moreover from x **have** $\mathfrak{C}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|x|\!)|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|\mathfrak{J}(\!|CId|\!)(\!|x|\!)|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
ultimately show $\mathfrak{C}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|x|\!)|\!) \in_\circ \mathcal{R}_\circ$ $(\mathfrak{F}(\!|ArrMap|\!))$
by (*simp add*: $\mathfrak{F}$.*ArrMap.vsv-vimageI2*)
qed
moreover have $\mathcal{R}_\circ$ $(\mathfrak{F}(\!|ArrMap|\!)) \in_\circ$ *Vset* α
by (*force intro*: *vrange-in-VsetI* $\mathfrak{F}$.*tm-cf-ArrMap-in-Vset*)
ultimately show *?thesis* **by** *auto*
qed
qed (*auto simp*: *cat-small-cs-intros*)
qed (*auto intro*: *cat-cs-intros*)

```
from assms(3)[where A=‹?R›, OF this] obtain P_O π_O
  where π_O: π_O : P_O <_{CF.∏} ?R : 𝔍⦇Obj⦈ ↦↦_{Cα} ℭ
  by clarsimp

interpret π_O: is-cat-obj-prod α ‹𝔍⦇Obj⦈› ?R ℭ P_O π_O by (rule π_O)

have P_O: P_O ∈₀ ℭ⦇Obj⦈ by (intro π_O.cat-cone-obj)

have ?L u ∈₀ ℭ⦇Obj⦈ if u ∈₀ 𝔍⦇Arr⦈ for u
proof–
  from that obtain a b where u : a ↦_𝔍 b by auto
  then show ?thesis
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
qed

have tm-cf-discrete: tm-cf-discrete α (𝔍⦇Arr⦈) ?L ℭ
proof(intro tm-cf-discreteI)
  show 𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇f⦈⦈ ∈₀ ℭ⦇Obj⦈ if f ∈₀ 𝔍⦇Arr⦈ for f
  proof–
    from that obtain a b where f : a ↦_𝔍 b by auto
    then show ?thesis
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed

  show (λu∈₀𝔍⦇Arr⦈. 𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇u⦈⦈) ∈₀ Vset α
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    show ℛ₀ (λu∈₀𝔍⦇Arr⦈. ?L u) ∈₀ Vset α
    proof–
      have ℛ₀ (λu∈₀𝔍⦇Arr⦈. ?L u) ⊆₀ ℛ₀ (𝔉⦇ObjMap⦈)
      proof(rule vrange-VLambda-vsubset)
        fix f assume f ∈₀ 𝔍⦇Arr⦈
        then obtain a b where f : a ↦_𝔍 b by auto
        then show 𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇f⦈⦈ ∈₀ ℛ₀ (𝔉⦇ObjMap⦈)
          by
            (
              cs-concl cs-shallow
                cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
            )
      qed
      moreover have ℛ₀ (𝔉⦇ObjMap⦈) ∈₀ Vset α
        by (auto intro: vrange-in-VsetI 𝔉.tm-cf-ObjMap-in-Vset)
      ultimately show ?thesis by auto
    qed
  qed (auto simp: cat-small-cs-intros)

  show (λi∈₀𝔍⦇Arr⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇i⦈⦈⦈) ∈₀ Vset α
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    show ℛ₀ (λi∈₀𝔍⦇Arr⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇i⦈⦈⦈) ∈₀ Vset α
    proof–
      have ℛ₀ (λi∈₀𝔍⦇Arr⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇i⦈⦈⦈) ⊆₀ ℛ₀ (𝔉⦇ArrMap⦈)
      proof(rule vrange-VLambda-vsubset)
        fix f assume f ∈₀ 𝔍⦇Arr⦈
        then obtain a b where f: f : a ↦_𝔍 b by auto
        then have 𝔍⦇CId⦈⦇b⦈ ∈₀ 𝒟₀ (𝔉⦇ArrMap⦈)
          by (auto intro: cat-cs-intros simp: cat-cs-simps)
        moreover from f have
          ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇𝔍⦇Cod⦈⦇f⦈⦈⦈ = 𝔉⦇ArrMap⦈⦇𝔍⦇CId⦈⦇b⦈⦈
```

```
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    ultimately show 𝔈(‖CId‖)(‖𝔉(‖ObjMap‖)(‖𝔍(‖Cod‖)(‖f‖)‖)‖) ∈∘ ℛ∘ (𝔉(‖ArrMap‖))
      by (simp add: 𝔉.ArrMap.vsv-vimageI2)
  qed
  moreover have ℛ∘ (𝔉(‖ArrMap‖)) ∈∘ Vset α
    by (force intro: vrange-in-VsetI 𝔉.tm-cf-ArrMap-in-Vset)
  ultimately show ?thesis by auto
 qed
qed (auto simp: cat-small-cs-intros)
qed (auto intro: cat-cs-intros)

from assms(4)[where A=‹?L›, OF this, simplified] obtain P_A π_A
  where π_A: π_A : P_A <_{CF.∏} ?L : 𝔍(‖Arr‖) ↦↦_{Cα} 𝔈
  by auto

interpret π_A: is-cat-obj-prod α ‹𝔍(‖Arr‖)› ?L 𝔈 P_A π_A by (rule π_A)

let ?F = ‹λu. 𝔉(‖ObjMap‖)(‖𝔍(‖Cod‖)(‖u‖)‖)› and ?f = ‹λu. π_O(‖NTMap‖)(‖𝔍(‖Cod‖)(‖u‖)‖)›
let ?π_O' = ‹ntcf-obj-prod-base 𝔈 (:_C (𝔍(‖Arr‖))(‖Obj‖)) ?F P_O ?f›

have π_O': ?π_O' :
  P_O <_{CF.cone} :→: (𝔍(‖Arr‖)) (λu. 𝔉(‖ObjMap‖)(‖𝔍(‖Cod‖)(‖u‖)‖)) 𝔈 :
  :_C (𝔍(‖Arr‖)) ↦↦_{Cα} 𝔈
  unfolding the-cat-discrete-components(1)
proof
  (
    intro
      tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone
      tm-cf-discrete
  )
  fix f assume f ∈∘ 𝔍(‖Arr‖)
  then obtain a b where f : a ↦_𝔍 b by auto
  then show π_O(‖NTMap‖)(‖𝔍(‖Cod‖)(‖f‖)‖) : P_O ↦_𝔈 𝔉(‖ObjMap‖)(‖𝔍(‖Cod‖)(‖f‖)‖)
    by
      (
        cs-concl cs-shallow
          cs-simp:
            the-cat-discrete-components(1) cat-discrete-cs-simps cat-cs-simps
          cs-intro: cat-cs-intros
      )
qed (intro P_O)

from π_A.cat-obj-prod-unique-cone'[OF π_O'] obtain f'
  where f': f' : P_O ↦_𝔈 P_A
    and π_O'-NTMap-app:
      ⋀j. j ∈∘ 𝔍(‖Arr‖) ⟹ ?π_O'(‖NTMap‖)(‖j‖) = π_A(‖NTMap‖)(‖j‖) ∘_{A𝔈} f'
    and unique-f':
      ⟦
        f'' : P_O ↦_𝔈 P_A;
        ⋀j. j ∈∘ 𝔍(‖Arr‖) ⟹ ?π_O'(‖NTMap‖)(‖j‖) = π_A(‖NTMap‖)(‖j‖) ∘_{A𝔈} f''
      ⟧ ⟹ f'' = f'
    for f''
  by metis

have π_O-NTMap-app-Cod:
  π_O(‖NTMap‖)(‖b‖) = π_A(‖NTMap‖)(‖f‖) ∘_{A𝔈} f' if f : a ↦_𝔍 b for f a b
proof–
  from that have f ∈∘ 𝔍(‖Arr‖) by auto
```

```
  from π_O'-NTMap-app[OF this] that show ?thesis
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps the-cat-discrete-components(1)
          cs-intro: cat-cs-intros
      )
qed

from this[symmetric] have π_A-NTMap-Comp-app:
  π_A(⁅NTMap⁆)(⁅f⁆) ∘_A𝔈 (f' ∘_A𝔈 q) = π_O(⁅NTMap⁆)(⁅b⁆) ∘_A𝔈 q
  if f : a ↦_𝔍 b and q : c ↦_𝔈 P_O for q f a b c
  using that f'
  by (intro 𝔉.HomCod.cat-assoc-helper)
    (
      cs-concl cs-shallow
        cs-simp:
          cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)
        cs-intro: cat-cs-intros
    )+

let ?g = ‹λu. 𝔉(⁅ArrMap⁆)(⁅u⁆) ∘_A𝔈 π_O(⁅NTMap⁆)(⁅𝔍(⁅Dom⁆)(⁅u⁆)⁆)›
let ?π_O'' = ‹ntcf-obj-prod-base 𝔈 (:_C (𝔍(⁅Arr⁆))(⁅Obj⁆)) ?F P_O ?g›

have π_O'': ?π_O'' : P_O <_CF.cone :→: (𝔍(⁅Arr⁆)) ?L 𝔈 : :_C (𝔍(⁅Arr⁆)) ↦↦_Cα 𝔈
  unfolding the-cat-discrete-components(1)
proof
  (
    intro
      tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone
      tm-cf-discrete
  )
  show 𝔉(⁅ArrMap⁆)(⁅f⁆) ∘_A𝔈 π_O(⁅NTMap⁆)(⁅𝔍(⁅Dom⁆)(⁅f⁆)⁆) : P_O ↦_𝔈 𝔉(⁅ObjMap⁆)(⁅𝔍(⁅Cod⁆)(⁅f⁆)⁆)
    if f ∈_∘ 𝔍(⁅Arr⁆) for f
  proof–
    from that obtain a b where f : a ↦_𝔍 b  by auto
    then show ?thesis
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps cat-discrete-cs-simps
              the-cat-discrete-components(1)
            cs-intro: cat-cs-intros
        )
  qed
qed (intro P_O)

from π_A.cat-obj-prod-unique-cone'[OF π_O''] obtain g'
  where g': g' : P_O ↦_𝔈 P_A
    and π_O''-NTMap-app:
      ⋀j. j ∈_∘ 𝔍(⁅Arr⁆) ⟹ ?π_O''(⁅NTMap⁆)(⁅j⁆) = π_A(⁅NTMap⁆)(⁅j⁆) ∘_A𝔈 g'
    and unique-g':
      ⟦
        g'' : P_O ↦_𝔈 P_A;
        ⋀j. j ∈_∘ 𝔍(⁅Arr⁆) ⟹ ?π_O''(⁅NTMap⁆)(⁅j⁆) = π_A(⁅NTMap⁆)(⁅j⁆) ∘_A𝔈 g''
      ⟧ ⟹ g'' = g'
    for g''
```

```
  by (metis (lifting))

have 𝔉(⌊ArrMap⌋)(⌊f⌋) ∘_A𝔈 π_O(⌊NTMap⌋)(⌊a⌋) = π_A(⌊NTMap⌋)(⌊f⌋) ∘_A𝔈 g'
  if f : a ↦_𝔍 b for f a b
proof–
  from that have f ∈_∘ 𝔍(⌊Arr⌋) by auto
  from π_O''-NTMap-app[OF this] that show ?thesis
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps the-cat-discrete-components(1)
          cs-intro: cat-cs-intros
      )
qed
then have π_O-NTMap-app-Dom:
  𝔉(⌊ArrMap⌋)(⌊f⌋) ∘_A𝔈 (π_O(⌊NTMap⌋)(⌊a⌋) ∘_A𝔈 q) =
    (π_A(⌊NTMap⌋)(⌊f⌋) ∘_A𝔈 g') ∘_A𝔈 q
  if f : a ↦_𝔍 b and q : c ↦_𝔈 P_O for q f a b c
  using that g'
  by (intro 𝔉.HomCod.cat-assoc-helper)
    (
      cs-concl
        cs-simp:
          cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)
        cs-intro: cat-cs-intros
    )

from assms(2)[OF f' g'] obtain E ε where ε:
  ε : E <_CF.eq (P_O,P_A,g',f') : ↑↑_C ↦↦_Cα 𝔈
  by clarsimp

interpret ε: is-cat-equalizer-2 α P_O P_A g' f' 𝔈 E ε by (rule ε)

define μ where μ =
  [(λi∈_∘𝔍(⌊Obj⌋). π_O(⌊NTMap⌋)(⌊i⌋) ∘_A𝔈 ε(⌊NTMap⌋)(⌊𝔞_PL2⌋)), cf-const 𝔍 𝔈 E, 𝔉, 𝔍, 𝔈]_∘

have μ-components:
  μ(⌊NTMap⌋) = (λi∈_∘𝔍(⌊Obj⌋). π_O(⌊NTMap⌋)(⌊i⌋) ∘_A𝔈 ε(⌊NTMap⌋)(⌊𝔞_PL2⌋))
  μ(⌊NTDom⌋) = cf-const 𝔍 𝔈 E
  μ(⌊NTCod⌋) = 𝔉
  μ(⌊NTDGDom⌋) = 𝔍
  μ(⌊NTDGCod⌋) = 𝔈
  unfolding μ-def nt-field-simps by (simp-all add: nat-omega-simps)

have [cat-cs-simps]:
  μ(⌊NTMap⌋)(⌊i⌋) = π_O(⌊NTMap⌋)(⌊i⌋) ∘_A𝔈 ε(⌊NTMap⌋)(⌊𝔞_PL2⌋) if i ∈_∘ 𝔍(⌊Obj⌋)
  for i
  using that unfolding μ-components by simp

have μ : E <_CF.lim 𝔉 : 𝔍 ↦↦_Cα 𝔈
proof(intro is-cat-limitI)

  show μ: μ : E <_CF.cone 𝔉 : 𝔍 ↦↦_Cα 𝔈
  proof(intro is-cat-coneI is-tm-ntcfI' is-ntcfI')
    show vfsequence μ unfolding μ-def by simp
    show vcard μ = 5_ℕ unfolding μ-def by (simp add: nat-omega-simps)
    show cf-const 𝔍 𝔈 E : 𝔍 ↦↦_Cα 𝔈
      by (cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros)
```

```
  show 𝔉 : 𝔍 ↦↦_CA 𝔈 by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show μ(|NTMap|)(|a|) : cf-const 𝔍 𝔈 E(|ObjMap|)(|a|) ↦_𝔈 𝔉(|ObjMap|)(|a|)
    if a ∈_∘ 𝔍(|Obj|) for a
    using that
    by
      (
        cs-concl
          cs-simp:
            cat-cs-simps
            cat-discrete-cs-simps
            cat-parallel-cs-simps
            the-cat-discrete-components(1)
          cs-intro: cat-cs-intros cat-lim-cs-intros cat-parallel-cs-intros
      )
  show
    μ(|NTMap|)(|b|) ∘_A𝔈 cf-const 𝔍 𝔈 E(|ArrMap|)(|f|) =
      𝔉(|ArrMap|)(|f|) ∘_A𝔈 μ(|NTMap|)(|a|)
    if f : a ↦_𝔍 b for a b f
    using that ε g′ f′
    by
      (
        cs-concl
          cs-simp:
            cat-parallel-cs-simps
            cat-cs-simps
            the-cat-discrete-components(1)
            π_O-NTMap-app-Cod
            π_O-NTMap-app-Dom
            ε.cat-eq-2-Comp-eq(1)
          cs-intro: cat-lim-cs-intros cat-cs-intros cat-parallel-cs-intros
      )

qed (auto simp: μ-components cat-cs-intros)

interpret μ: is-cat-cone α E 𝔍 𝔈 𝔉 μ by (rule μ)

show ∃!f′. f′ : r′ ↦_𝔈 E ∧ u′ = μ ∙_NTCF ntcf-const 𝔍 𝔈 f′
  if u′ : r′ <_CF.cone 𝔉 : 𝔍 ↦↦_Cα 𝔈 for u′ r′
proof–

  interpret u′: is-cat-cone α r′ 𝔍 𝔈 𝔉 u′ by (rule that)

  let ?u′ = ‹λj. u′(|NTMap|)(|j|)›
  let ?π′ = ‹ntcf-obj-prod-base 𝔈 (𝔍(|Obj|)) ?R r′ ?u′›

  have π′-NTMap-app: ?π′(|NTMap|)(|j|) = u′(|NTMap|)(|j|) if j ∈_∘ 𝔍(|Obj|) for j
    using that
    unfolding ntcf-obj-prod-base-components the-cat-discrete-components
    by auto

  have π′: ?π′ : r′ <_CF.cone :→: (𝔍(|Obj|)) ?R 𝔈 : :_C (𝔍(|Obj|)) ↦↦_Cα 𝔈
    unfolding the-cat-discrete-components(1)
  proof(intro tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone)
    show tm-cf-discrete α (𝔍(|Obj|)) ?R 𝔈
    proof(intro tm-cf-discreteI)
      show 𝔉(|ObjMap|)(|i|) ∈_∘ 𝔈(|Obj|) if i ∈_∘ 𝔍(|Obj|) for i
        by (cs-concl cs-simp: cat-cs-simps cs-intro: that cat-cs-intros)
      show category α 𝔈 by (auto intro: cat-cs-intros)
```

```
    from 𝔉.tm-cf-ObjMap-in-Vset show (λx∈₀𝔍⦇Obj⦈. 𝔉⦇ObjMap⦈⦇x⦈) ∈₀ Vset α
      by (auto simp: 𝔉.cf-ObjMap-vdomain)
    show (λi∈₀𝔍⦇Obj⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇i⦈⦈) ∈₀ Vset α
    proof(rule vbrelation.vbrelation-Limit-in-VsetI)
      have ℛ₀ (λi∈₀𝔍⦇Obj⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇i⦈⦈) ⊆₀ ℛ₀ (𝔉⦇ArrMap⦈)
      proof(intro vsubsetI)
        fix x assume x ∈₀ ℛ₀ (λi∈₀𝔍⦇Obj⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇i⦈⦈)
        then obtain i where i: i ∈₀ 𝔍⦇Obj⦈
          and x-def: x = ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇i⦈⦈
          by auto
        from i have x = 𝔉⦇ArrMap⦈⦇𝔍⦇CId⦈⦇i⦈⦈
          by (simp add: x-def 𝔉.cf-ObjMap-CId)
        moreover from i have 𝔍⦇CId⦈⦇i⦈ ∈₀ 𝒟₀ (𝔉⦇ArrMap⦈)
          by
            (
              cs-concl cs-shallow
                cs-simp: cat-cs-simps cs-intro: cat-cs-intros
            )
        ultimately show x ∈₀ ℛ₀ (𝔉⦇ArrMap⦈)
          by (auto intro: 𝔉.ArrMap.vsv-vimageI2)
      qed
      then show ℛ₀ (λi∈₀𝔍⦇Obj⦈. ℭ⦇CId⦈⦇𝔉⦇ObjMap⦈⦇i⦈⦈) ∈₀ Vset α
        by
          (
            auto simp:
              𝔉.tm-cf-ArrMap-in-Vset vrange-in-VsetI vsubset-in-VsetI
          )
    qed (auto intro: 𝔉.HomDom.tiny-cat-Obj-in-Vset)
  qed
  show u′⦇NTMap⦈⦇j⦈ : r′ ↦ℭ 𝔉⦇ObjMap⦈⦇j⦈ if j ∈₀ 𝔍⦇Obj⦈ for j
    using that
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
qed (auto simp: cat-cs-intros)

from πO.cat-obj-prod-unique-cone′[OF this] obtain h′
  where h′: h′ : r′ ↦ℭ P_O
    and π′-NTMap-app′:
      ⋀j. j ∈₀ (𝔍⦇Obj⦈) ⟹ ?π′⦇NTMap⦈⦇j⦈ = πO⦇NTMap⦈⦇j⦈ ∘_Aℭ h′
    and unique-h′: ⋀h″.
      ⟦
        h″ : r′ ↦ℭ P_O;
        ⋀j. j ∈₀ (𝔍⦇Obj⦈) ⟹ ?π′⦇NTMap⦈⦇j⦈ = πO⦇NTMap⦈⦇j⦈ ∘_Aℭ h″
      ⟧ ⟹ h″ = h′
  by metis

interpret π′:
  is-cat-cone α r′ ‹:_C (𝔍⦇Obj⦈)› ℭ ‹:→: (𝔍⦇Obj⦈) (app (𝔉⦇ObjMap⦈)) ℭ› ?π′
  by (rule π′)

let ?u″ = ‹λu. u′⦇NTMap⦈⦇𝔍⦇Cod⦈⦇u⦈⦈›
let ?π″ = ‹ntcf-obj-prod-base ℭ (𝔍⦇Arr⦈) ?L r′ ?u″›

have π″-NTMap-app: ?π″⦇NTMap⦈⦇f⦈ = u′⦇NTMap⦈⦇b⦈
  if f : a ↦𝔍 b for f a b
  using that
  unfolding ntcf-obj-prod-base-components the-cat-discrete-components
  by
    (
```

```
    cs-concl cs-shallow
      cs-simp: V-cs-simps cat-cs-simps cs-intro: cat-cs-intros
  )

have π'': ?π'' : r' <CF.cone :→: (𝔍(|Arr|)) ?L 𝔈 : :C (𝔍(|Arr|)) ↦↦Cα 𝔈
  unfolding the-cat-discrete-components(1)
proof
  (
    intro
      tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone
      tm-cf-discrete
  )
  fix f assume f ∈∘ 𝔍(|Arr|)
  then obtain a b where f : a ↦𝔍 b by auto
  then show u'(|NTMap|)(𝔍(|Cod|)(|f|)) : r' ↦𝔈 𝔉(|ObjMap|)(𝔍(|Cod|)(|f|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros
      )
qed (simp add: cat-cs-intros)

from πA.cat-obj-prod-unique-cone'[OF this] obtain h''
  where h'': h'' : r' ↦𝔈 PA
    and π''-NTMap-app':
      ⋀j. j ∈∘ 𝔍(|Arr|) ⟹ ?π''(|NTMap|)(|j|) = πA(|NTMap|)(|j|) ∘A𝔈 h''
    and unique-h'': ⋀h'''.
      ⟦
        h''' : r' ↦𝔈 PA;
        ⋀j. j ∈∘ 𝔍(|Arr|) ⟹ ?π''(|NTMap|)(|j|) = πA(|NTMap|)(|j|) ∘A𝔈 h'''
      ⟧ ⟹ h''' = h''
  by metis

interpret π'': is-cat-cone α r' ‹:C (𝔍(|Arr|))› 𝔈 ‹:→: (𝔍(|Arr|)) ?L 𝔈› ?π''
  by (rule π'')

have g'h'-f'h': g' ∘A𝔈 h' = f' ∘A𝔈 h'
proof–

  from g' h' have g'h': g' ∘A𝔈 h' : r' ↦𝔈 PA
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from f' h' have f'h': f' ∘A𝔈 h' : r' ↦𝔈 PA
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  have ?π''(|NTMap|)(|f|) = πA(|NTMap|)(|f|) ∘A𝔈 (g' ∘A𝔈 h')
    if f ∈∘ 𝔍(|Arr|) for f
  proof–
    from that obtain a b where f: f : a ↦𝔍 b by auto
    then have ?π''(|NTMap|)(|f|) = u'(|NTMap|)(|b|)
      by (cs-concl cs-simp: π''-NTMap-app cat-cs-simps)
    also from f have ... = 𝔉(|ArrMap|)(|f|) ∘A𝔈 ?π'(|NTMap|)(|a|)
      by
        (
          cs-concl
            cs-simp: π'-NTMap-app cat-cs-simps cat-lim-cs-simps
            cs-intro: cat-cs-intros
        )
    also from f g' h' have ... = πA(|NTMap|)(|f|) ∘A𝔈 (g' ∘A𝔈 h')
```

```
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-discrete-cs-simps
              the-cat-discrete-components(1)
              π′-NTMap-app′
              π_O-NTMap-app-Dom
            cs-intro: cat-cs-intro
        )
    finally show ?thesis by simp
  qed

  from unique-h′′[OF g′h′ this, simplified] have g′h′-h′′:
    g′ ∘_Aℭ h′ = h′′.
  have ?π′′(|NTMap|)(|f|) = π_A(|NTMap|)(|f|) ∘_Aℭ (f′ ∘_Aℭ h′)
    if f ∈_∘ ℑ(|Arr|) for f
  proof–
    from that obtain a b where f: f : a ↦_ℑ b by auto
    then have ?π′′(|NTMap|)(|f|) = u′(|NTMap|)(|b|)
      by (cs-concl cs-simp: π′′-NTMap-app cat-cs-simps)
    also from f have ... = ?π′(|NTMap|)(|b|)
      by
        (
          cs-concl cs-shallow
            cs-simp: π′-NTMap-app cs-intro: cat-cs-intro
        )
    also from f have ... = π_O(|NTMap|)(|b|) ∘_Aℭ h′
      by
        (
          cs-concl cs-shallow
            cs-simp: π′-NTMap-app′ cs-intro: cat-cs-intro
        )
    also from f g′ h′ have ... = (π_A(|NTMap|)(|f|) ∘_Aℭ f′) ∘_Aℭ h′
      by
        (
          cs-concl cs-shallow
            cs-simp: π_O-NTMap-app-Cod cs-intro: cat-cs-intro
        )
    also from that f′ h′ have ... = π_A(|NTMap|)(|f|) ∘_Aℭ (f′ ∘_Aℭ h′)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps the-cat-discrete-components(1)
            cs-intro: cat-cs-intro
         )
    finally show ?thesis by simp
  qed
  from unique-h′′[OF f′h′ this, simplified] have f′h′-h′′:
    f′ ∘_Aℭ h′ = h′′.
  from g′h′-h′′ f′h′-h′′ show ?thesis by simp
qed

let ?II = ‹↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL›
  and ?II-II = ‹↑↑→↑↑_CF ℭ 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL P_O P_A g′ f′›

define ε′ where ε′ =
```

```
  [
    (λf∈∘ ?II(|Obj|). (f = 𝔞_PL2 ? h' : (f' ∘_A𝔈 h'))),
    cf-const ?II 𝔈 r',
    ?II-II,
    ?II,
    𝔈
  ]∘

have ε'-components:
  ε'(|NTMap|) = (λf∈∘ ?II(|Obj|). (f = 𝔞_PL2 ? h' : (f' ∘_A𝔈 h')))
  ε'(|NTDom|) = cf-const ?II 𝔈 r'
  ε'(|NTCod|) = ?II-II
  ε'(|NTDGDom|) = ?II
  ε'(|NTDGCod|) = 𝔈
  unfolding ε'-def nt-field-simps by (simp-all add: nat-omega-simps)

have ε'-NTMap-app-I2: ε'(|NTMap|)(|x|) = h' if x = 𝔞_PL2 for x
proof–
  have x ∈∘ ?II(|Obj|)
    unfolding that by (cs-concl cs-intro: cat-parallel-cs-intros)
  then show ?thesis unfolding ε'-components that by simp
qed

have ε'-NTMap-app-sI2: ε'(|NTMap|)(|x|) = f' ∘_A𝔈 h' if x = 𝔟_PL2 for x
proof–
  have x ∈∘ ?II(|Obj|)
    unfolding that by (cs-concl cs-shallow cs-intro: cat-parallel-cs-intros)
  with ε.cat-parallel-𝔞𝔟 show ?thesis
    unfolding ε'-components by (cs-concl cs-simp: V-cs-simps that)
qed

interpret par: cf-parallel-2 α 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL P_O P_A g' f' 𝔈
  by (intro cf-parallel-2I cat-parallel-2I)
    (simp-all add: cat-cs-intros cat-parallel-cs-intros)

have ε' : r' <_CF.cone ?II-II : ?II ↦↦_Cα 𝔈
proof(intro is-cat-coneI is-tm-ntcfI' is-ntcfI')
  show vfsequence ε' unfolding ε'-def by auto
  show vcard ε' = 5_ℕ unfolding ε'-def by (simp add: nat-omega-simps)
  from h' show cf-const (?II) 𝔈 r' : ?II ↦↦_Cα 𝔈
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show ?II-II : ?II ↦↦_Cα 𝔈
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-parallel-cs-simps cs-intro: cat-cs-intros
      )
  from h' show ε'(|NTMap|)(|a|) :
    cf-const ?II 𝔈 r'(|ObjMap|)(|a|) ↦_𝔈 ?II-II(|ObjMap|)(|a|)
    if a ∈∘ ?II(|Obj|) for a
    using that
    by (elim the-cat-parallel-2-ObjE; simp only:)
      (
        cs-concl
          cs-simp:
            ε'-NTMap-app-I2 ε'-NTMap-app-sI2
            cat-cs-simps cat-parallel-cs-simps
          cs-intro: cat-cs-intros cat-parallel-cs-intros
```

```
    )
  from h' f' g'h'-f'h' show
    ε'(|NTMap|)(|b|) ∘_A𝔈 cf-const ?II 𝔈 r'(|ArrMap|)(|f|) =
      ?II-II(|ArrMap|)(|f|) ∘_A𝔈 ε'(|NTMap|)(|a|)
      if f : a ↦_?II b for a b f
      using that
      by (elim ε.the-cat-parallel-2-is-arrE; simp only:)
        (
          cs-concl
            cs-intro: cat-cs-intros cat-parallel-cs-intros
            cs-simp:
              cat-cs-simps
              cat-parallel-cs-simps
              ε'-NTMap-app-I2
              ε'-NTMap-app-sI2
        )+
  qed
    (
      simp add: ε'-components |
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-lim-cs-intros cat-cs-intros cat-small-cs-intros
    )+
from ε.cat-eq-2-unique-cone[OF this] obtain t'
  where t': t' : r' ↦_𝔈 E
    and ε'-NTMap-app: ε'(|NTMap|)(|𝔞_PL2|) = ε(|NTMap|)(|𝔞_PL2|) ∘_A𝔈 t'
    and unique-t':
      [[ t'' : r' ↦_𝔈 E; ε'(|NTMap|)(|𝔞_PL2|) = ε(|NTMap|)(|𝔞_PL2|) ∘_A𝔈 t'' ]] ⟹
        t'' = t'
    for t''
  by metis

show ∃!f'. f' : r' ↦_𝔈 E ∧ u' = μ ·_NTCF ntcf-const 𝔍 𝔈 f'
proof(intro ex1I conjI; (elim conjE)?, (rule t')?)
  show [symmetric, cat-cs-simps]: u' = μ ·_NTCF ntcf-const 𝔍 𝔈 t'
  proof(rule ntcf-eqI[OF u'.is-ntcf-axioms])
    from t' show
      μ ·_NTCF ntcf-const 𝔍 𝔈 t' : cf-const 𝔍 𝔈 r' ↦_CF 𝔉 : 𝔍 ↦↦_Cα 𝔈
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show u'(|NTMap|) = (μ ·_NTCF ntcf-const 𝔍 𝔈 t')(|NTMap|)
    proof(rule vsv-eqI)
      show vsv ((μ ·_NTCF ntcf-const 𝔍 𝔈 t')(|NTMap|))
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      from t' show
        𝒟_∘ (u'(|NTMap|)) = 𝒟_∘ ((μ ·_NTCF ntcf-const 𝔍 𝔈 t')(|NTMap|))
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros
          )
      show u'(|NTMap|)(|a|) = (μ ·_NTCF ntcf-const 𝔍 𝔈 t')(|NTMap|)(|a|)
        if a ∈_∘ 𝒟_∘ (u'(|NTMap|)) for a
      proof–
        from that have a ∈_∘ 𝔍(|Obj|)
          by (cs-prems cs-shallow cs-simp: cat-cs-simps)
        with t' show u'(|NTMap|)(|a|) = (μ ·_NTCF ntcf-const 𝔍 𝔈 t')(|NTMap|)(|a|)
          by
            (
```

```
            cs-concl
              cs-simp:
                cat-cs-simps
                π′-NTMap-app
                cat-parallel-cs-simps
                the-cat-discrete-components(1)
                ε′-NTMap-app[symmetric]
                ε′-NTMap-app-I2
                π′-NTMap-app′[symmetric]
              cs-intro: cat-cs-intros cat-parallel-cs-intros
          )
        qed
      qed auto
    qed simp-all

    fix t′′ assume prems′: t′′ : r′ ↦_ℭ E u′ = μ ⋆_NTCF ntcf-const ℑ ℭ t′′
    then have u′-NTMap-app-x:
      u′(|NTMap|)(|x|) = (μ ⋆_NTCF ntcf-const ℑ ℭ t′′)(|NTMap|)(|x|)
      for x
      by simp
    have ?π′(|NTMap|)(|j|) = π_O(|NTMap|)(|j|) ∘_Aℭ (ε(|NTMap|)(|𝔞_PL2|) ∘_Aℭ t′′)
      if j ∈_∘ ℑ(|Obj|) for j
      using u′-NTMap-app-x[of j] prems′(1) that
      by
        (
          cs-prems
            cs-simp:
              cat-cs-simps
              cat-discrete-cs-simps
              cat-parallel-cs-simps
              the-cat-discrete-components(1)
            cs-intro: cat-cs-intros cat-parallel-cs-intros
        )
        (simp add: π′-NTMap-app[OF that, symmetric])
    moreover from prems′(1) have ε(|NTMap|)(|𝔞_PL2|) ∘_Aℭ t′′ : r′ ↦_ℭ P_O
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-parallel-cs-simps
            cs-intro: cat-cs-intros cat-parallel-cs-intros
        )
    ultimately have [cat-cs-simps]: ε(|NTMap|)(|𝔞_PL2|) ∘_Aℭ t′′ = h′
      by (intro unique-h′) simp
    show t′′ = t′
      by (rule unique-t′, intro prems′(1))
        (cs-concl cs-shallow cs-simp: ε′-NTMap-app-I2 cat-cs-simps)
    qed
  qed

  qed

  then show ?thesis using that by clarsimp

qed

lemma cat-colimit-of-cat-prod-obj-and-cat-coequalizer:
  — See Theorem 1 in Chapter V-2 in [9].
  assumes 𝔉 : ℑ ↦↦_Cat_Cα ℭ
```

and $\bigwedge \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}.\ [\![\ \mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a};\ \mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}\ ]\!] \Longrightarrow$
$\exists E\ \varepsilon.\ \varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge A.$ *tm-cf-discrete* α $(\mathfrak{J}(\!|Obj|\!))$ A $\mathfrak{C} \Longrightarrow$
$\exists P\ \pi.\ \pi : A >_{CF.\coprod} P : \mathfrak{J}(\!|Obj|\!) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge A.$ *tm-cf-discrete* α $(\mathfrak{J}(\!|Arr|\!))$ A $\mathfrak{C} \Longrightarrow$
$\exists P\ \pi.\ \pi : A >_{CF.\coprod} P : \mathfrak{J}(\!|Arr|\!) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains $r\ u$ **where** $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
have $\exists E\ \varepsilon.\ \varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
if $\mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ $\mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ **for** $\mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$
proof–
from *assms*(*2*)[*OF that*(*1,2*)] **obtain** $E\ \varepsilon$
where ε: $\varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by *clarsimp*
interpret ε: *is-cat-coequalizer-2* α $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{C}$ E ε **by** (*rule* ε)
from ε.*is-cat-equalizer-2-op*[*unfolded cat-op-simps*] **show** *?thesis* **by** *auto*
qed
moreover have $\exists P\ \pi.\ \pi : P <_{CF.\prod} A : \mathfrak{J}(\!|Obj|\!) \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
if *tm-cf-discrete* α $(\mathfrak{J}(\!|Obj|\!))$ A (*op-cat* $\mathfrak{C}$) **for** A
proof–
interpret *tm-cf-discrete* α $\langle\mathfrak{J}(\!|Obj|\!)\rangle$ A $\langle$*op-cat* $\mathfrak{C}\rangle$ **by** (*rule that*)
from *assms*(*3*)[*OF tm-cf-discrete-op*[*unfolded cat-op-simps*]] **obtain** $P\ \pi$
where π: $\pi : A >_{CF.\coprod} P : \mathfrak{J}(\!|Obj|\!) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by *clarsimp*
interpret π: *is-cat-obj-coprod* α $\langle\mathfrak{J}(\!|Obj|\!)\rangle$ A $\mathfrak{C}$ P π **by** (*rule* π)
from π.*is-cat-obj-prod-op* **show** *?thesis* **by** *auto*
qed
moreover have $\exists P\ \pi.\ \pi : P <_{CF.\prod} A : \mathfrak{J}(\!|Arr|\!) \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
if *tm-cf-discrete* α $(\mathfrak{J}(\!|Arr|\!))$ A (*op-cat* $\mathfrak{C}$) **for** A
proof–
interpret *tm-cf-discrete* α $\langle\mathfrak{J}(\!|Arr|\!)\rangle$ A $\langle$*op-cat* $\mathfrak{C}\rangle$ **by** (*rule that*)
from *assms*(*4*)[*OF tm-cf-discrete-op*[*unfolded cat-op-simps*]] **obtain** $P\ \pi$
where π: $\pi : A >_{CF.\coprod} P : \mathfrak{J}(\!|Arr|\!) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by *clarsimp*
interpret π: *is-cat-obj-coprod* α $\langle\mathfrak{J}(\!|Arr|\!)\rangle$ A $\mathfrak{C}$ P π **by** (*rule* π)
from π.*is-cat-obj-prod-op* **show** *?thesis* **by** *auto*
qed
ultimately obtain $u\ r$ **where** u:
$u : r <_{CF.lim}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{J} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by
(
rule cat-limit-of-cat-prod-obj-and-cat-equalizer[
OF $\mathfrak{F}$.*is-tm-functor-op*, *unfolded cat-op-simps*
]
)
interpret u: *is-cat-limit* α $\langle$*op-cat* $\mathfrak{J}\rangle$ $\langle$*op-cat* $\mathfrak{C}\rangle$ $\langle$*op-cf* $\mathfrak{F}\rangle$ r u **by** (*rule* u)
from u.*is-cat-colimit-op*[*unfolded cat-op-simps*] *that* **show** *?thesis* **by** *simp*
qed

10.2 Small-complete and small-cocomplete category

10.2.1 Definition and elementary properties

locale *cat-small-complete* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
assumes *cat-small-complete*:
$\bigwedge \mathfrak{F}\ \mathfrak{J}.\ \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \Longrightarrow \exists u\ r.\ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

locale *cat-small-cocomplete* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *cat-small-cocomplete*:
    $\bigwedge \mathfrak{F}\ \mathfrak{J}.\ \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \Longrightarrow \exists u\ r.\ u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

Rules.

mk-ide rf *cat-small-complete-def*[*unfolded cat-small-complete-axioms-def*]
  |*intro cat-small-completeI*|
  |*dest cat-small-completeD*[*dest*]|
  |*elim cat-small-completeE*[*elim*]|

lemma *cat-small-completeE'*[*elim*]:
  assumes *cat-small-complete* α $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
  obtains *u r* **where** $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  using *assms* **by** *auto*

mk-ide rf *cat-small-cocomplete-def*[*unfolded cat-small-cocomplete-axioms-def*]
  |*intro cat-small-cocompleteI*|
  |*dest cat-small-cocompleteD*[*dest*]|
  |*elim cat-small-cocompleteE*[*elim*]|

lemma *cat-small-cocompleteE'*[*elim*]:
  assumes *cat-small-cocomplete* α $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
  obtains *u r* **where** $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  using *assms* **by** *auto*

10.2.2 Duality

lemma (**in** *cat-small-complete*) *cat-small-cocomplete-op*[*cat-op-intros*]:
  cat-small-cocomplete α (*op-cat* $\mathfrak{C}$)
proof(*intro cat-small-cocompleteI*)
  fix $\mathfrak{F}$ $\mathfrak{J}$ **assume** $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha}$ *op-cat* $\mathfrak{C}$
  then interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ ‹*op-cat* $\mathfrak{C}$› $\mathfrak{F}$ **.**
  from *cat-small-complete*[*OF* $\mathfrak{F}$.*is-tm-functor-op*[*unfolded cat-op-simps*]]
  obtain *u r* **where** *u*: $u : r <_{CF.lim}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by *auto*
  then interpret *u*: *is-cat-limit* α ‹*op-cat* $\mathfrak{J}$› $\mathfrak{C}$ ‹*op-cf* $\mathfrak{F}$› *r u* **.**
  from *u.is-cat-colimit-op*[*unfolded cat-op-simps*] **show**
    $\exists u\ r.\ u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
    by *auto*
qed (*auto intro*: *cat-cs-intros*)

lemmas [*cat-op-intros*] = *cat-small-complete.cat-small-cocomplete-op*

lemma (**in** *cat-small-cocomplete*) *cat-small-complete-op*[*cat-op-intros*]:
  cat-small-complete α (*op-cat* $\mathfrak{C}$)
proof(*intro cat-small-completeI*)
  fix $\mathfrak{F}$ $\mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha}$ *op-cat* $\mathfrak{C}$
  then interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ ‹*op-cat* $\mathfrak{C}$› $\mathfrak{F}$ **.**
  from *cat-small-cocomplete*[*OF* $\mathfrak{F}$.*is-tm-functor-op*[*unfolded cat-op-simps*]]
  obtain *u r* **where** *u*: *u* : *op-cf* $\mathfrak{F} >_{CF.colim} r$: *op-cat* $\mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by *auto*
  interpret *u*: *is-cat-colimit* α ‹*op-cat* $\mathfrak{J}$› $\mathfrak{C}$ ‹*op-cf* $\mathfrak{F}$› *r u* **by** (*rule u*)
  from *u.is-cat-limit-op*[*unfolded cat-op-simps*] **show**
    $\exists u\ r.\ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
    by *auto*
qed (*auto intro*: *cat-cs-intros*)

lemmas [*cat-op-intros*] = *cat-small-cocomplete.cat-small-complete-op*

10.2.3 A category with equalizers and small products is small-complete

lemma (**in** *category*) *cat-small-complete-if-eq-and-obj-prod*:
— See Corollary 2 in Chapter V-2 in [9]
assumes $\bigwedge \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}.\ [\![\ \mathfrak{f} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b};\ \mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}\ ]\!] \Longrightarrow$
$\exists E\ \varepsilon.\ \varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge A\ I.$ *tm-cf-discrete* $\alpha\ I\ A\ \mathfrak{C} \Longrightarrow \exists P\ \pi.\ \pi : P <_{CF.\prod} A : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *cat-small-complete* $\alpha\ \mathfrak{C}$
proof(*intro cat-small-completeI*)
fix $\mathfrak{F}\ \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
then interpret $\mathfrak{F}$: *is-tm-functor* $\alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}$ **.**
show $\exists u\ r.\ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*rule cat-limit-of-cat-prod-obj-and-cat-equalizer*[*OF prems assms*(*1*)])
(*auto intro*: *assms*(*2*))
qed (*auto simp*: *cat-cs-intros*)

lemma (**in** *category*) *cat-small-cocomplete-if-eq-and-obj-prod*:
assumes $\bigwedge \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}.\ [\![\ \mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a};\ \mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}\ ]\!] \Longrightarrow$
$\exists E\ \varepsilon.\ \varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge A\ I.$ *tm-cf-discrete* $\alpha\ I\ A\ \mathfrak{C} \Longrightarrow \exists P\ \pi.\ \pi : A >_{CF.\coprod} P : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *cat-small-cocomplete* $\alpha\ \mathfrak{C}$
proof–
have $\exists E\ \varepsilon.\ \varepsilon : E <_{CF.eq} (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
if $\mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ **and** $\mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ **for** $\mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$
proof–
from *assms*(*1*)[*OF that*] **obtain** $\varepsilon\ E$ **where**
ε: $\varepsilon : (\mathfrak{a},\mathfrak{b},\mathfrak{g},\mathfrak{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by *clarsimp*
interpret ε: *is-cat-coequalizer-2* $\alpha\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{C}\ E\ \varepsilon$ **by** (*rule* ε)
from ε.*is-cat-equalizer-2-op* **show** *?thesis* **by** *auto*
qed
moreover have $\exists P\ \pi.\ \pi : P <_{CF.\prod} A : I \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
if *tm-cf-discrete* $\alpha\ I\ A$ (*op-cat* $\mathfrak{C}$) **for** $A\ I$
proof–
interpret *tm-cf-discrete* $\alpha\ I\ A$ ‹*op-cat* $\mathfrak{C}$› **by** (*rule that*)
from *assms*(*2*)[*OF tm-cf-discrete-op*[*unfolded cat-op-simps*]] **obtain** $P\ \pi$
where π: $\pi : A >_{CF.\coprod} P : I \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by *auto*
interpret π: *is-cat-obj-coprod* $\alpha\ I\ A\ \mathfrak{C}\ P\ \pi$ **by** (*rule* π)
from π.*is-cat-obj-prod-op* **show** *?thesis* **by** *auto*
qed
ultimately interpret *cat-small-complete* α ‹*op-cat* $\mathfrak{C}$›
by
(
rule category.cat-small-complete-if-eq-and-obj-prod[
OF category-op, *unfolded cat-op-simps*
]
)
show *?thesis* **by** (*rule cat-small-cocomplete-op*[*unfolded cat-op-simps*])
qed

10.2.4 Existence of the initial and terminal objects in small-complete and small-cocomplete categories

lemma (**in** *cat-small-complete*) *cat-sc-ex-obj-initial*:
— See Theorem 1 in Chapter V-6 in [9].
assumes $A \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $A \in_{\circ} Vset\ \alpha$
and $\bigwedge c.\ c \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \exists f\ a.\ a \in_{\circ} A \wedge f : a \mapsto_{\mathfrak{C}} c$

```
  obtains z where obj-initial 𝔈 z
proof–

  interpret tcd: tm-cf-discrete α A id 𝔈
  proof(intro tm-cf-discreteI)
    show (λi∈∘A. 𝔈⦇CId⦈⦇id i⦈) ∈∘ Vset α
      unfolding id-def
    proof(rule vbrelation.vbrelation-Limit-in-VsetI)
      from assms(1) have A ⊆∘ 𝒟∘ (𝔈⦇CId⦈) by (simp add: cat-CId-vdomain)
      then have ℛ∘ (VLambda A (app (𝔈⦇CId⦈))) = 𝔈⦇CId⦈ ‘∘ A by auto
      moreover have (⋃∘a∈∘A. ⋃∘b∈∘A. Hom 𝔈 a b) ∈∘ Vset α
        by (rule cat-Hom-vifunion-in-Vset[OF assms(1,1,2,2)])
      moreover have 𝔈⦇CId⦈ ‘∘ A ⊆∘ (⋃∘a∈∘A. ⋃∘b∈∘A. Hom 𝔈 a b)
      proof(intro vsubsetI)
        fix f assume f ∈∘ 𝔈⦇CId⦈ ‘∘ A
        then obtain a where a: a ∈∘ A and f-def: f = 𝔈⦇CId⦈⦇a⦈ by auto
        from assms(1) a show f ∈∘ (⋃∘a∈∘A. ⋃∘b∈∘A. Hom 𝔈 a b)
          unfolding f-def by (intro vifunionI) (auto simp: cat-CId-is-arr)
      qed
      ultimately show ℛ∘ (VLambda A (app (𝔈⦇CId⦈))) ∈∘ Vset α by auto
    qed (simp-all add: assms(2))
  qed
    (
      use assms in
        ‹auto simp: assms(2) Limit-vid-on-in-Vset intro: cat-cs-intros›
    )

  have tcd: :→: A id 𝔈 : :C A ↦↦→C.tmα 𝔈
    by
      (
        cs-concl cs-shallow cs-intro:
          cat-small-cs-intros
          cat-cs-intros
          cat-small-discrete-cs-intros
          cat-discrete-cs-intros
      )
  from cat-small-complete[OF this] obtain π w
    where π : w <CF.lim :→: A id 𝔈 : :C A ↦↦→Cα 𝔈
    by auto
  then interpret π: is-cat-obj-prod α A id 𝔈 w π
    by (intro is-cat-obj-prodI tcd.cf-discrete-axioms)

  let ?ww = ‹Hom 𝔈 w w›

  have CId-w: 𝔈⦇CId⦈⦇w⦈ ∈∘ ?ww
    by (cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros)
  then have ww-neq-vempty: ?ww ≠ 0 by force

  have wd: ∃h. h : w ↦𝔈 d if d ∈∘ 𝔈⦇Obj⦈ for d
  proof–
    from assms(3)[OF that] obtain g a where a: a ∈∘ A and g: g : a ↦𝔈 d
      by clarsimp
    from π.ntcf-NTMap-is-arr[unfolded the-cat-discrete-components, OF a] a g
    have π⦇NTMap⦈⦇a⦈ : w ↦𝔈 a
      by
        (
          cs-prems cs-shallow cs-simp:
            id-apply the-cat-discrete-components(1)
```

```
      cat-discrete-cs-simps cat-cs-simps
    )
  with g have g ∘_A𝔈 π(‖NTMap‖)(‖a‖) : w ↦_𝔈 d
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  then show ?thesis by (intro exI)
qed

have cf-parallel α (𝔞_PL ?ww) (𝔟_PL ?ww) ?ww w w (vid-on ?ww) 𝔈
  by (intro cat-cf-parallel-𝔞𝔟 π.cat-cone-obj cat-Hom-in-Vset) simp-all
then have ⇑→⇑_CF 𝔈 (𝔞_PL ?ww) (𝔟_PL ?ww) ?ww w w (vid-on ?ww) :
  ⇑_C (𝔞_PL ?ww) (𝔟_PL ?ww) ?ww ↦↦_C.tmα 𝔈
  by (intro cf-parallel.cf-parallel-the-cf-parallel-is-tm-functor)
from cat-small-complete[OF this] obtain ε v where ε: ε :
  v <_CF.lim ⇑→⇑_CF 𝔈 (𝔞_PL ?ww) (𝔟_PL ?ww) ?ww w w (vid-on ?ww) :
  ⇑_C (𝔞_PL ?ww) (𝔟_PL ?ww) ?ww ↦↦_Cα 𝔈
  by clarsimp
from is-cat-equalizerI[
    OF
      this
      -
      cat-Hom-in-Vset[OF π.cat-cone-obj π.cat-cone-obj]
      ww-neq-vempty
    ]
interpret ε: is-cat-equalizer α w w ?ww ‹vid-on ?ww› 𝔈 v ε by auto
note ε-is-monic-arr =
  is-cat-equalizer.cat-eq-is-monic-arr[OF ε.is-cat-equalizer-axioms]
note ε-is-monic-arrD = is-monic-arrD[OF ε-is-monic-arr]

show ?thesis
proof(rule, intro obj-initialI)

  show v ∈_∘ 𝔈(‖Obj‖) by (rule ε.cat-cone-obj)
  then have CId-v: 𝔈(‖CId‖)(‖v‖) : v ↦_𝔈 v
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)

  fix d assume prems: d ∈_∘ 𝔈(‖Obj‖)
  from wd[OF prems] obtain h where h: h : w ↦_𝔈 d by auto

  show ∃!f. f : v ↦_𝔈 d
  proof(rule ex1I)
    define f where f = h ∘_A𝔈 ε(‖NTMap‖)(‖𝔞_PL ?ww‖)
    from ε-is-monic-arrD(1) h show f: f : v ↦_𝔈 d
      unfolding f-def by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    fix g assume prems': g : v ↦_𝔈 d
    have cf-parallel-2 α 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL v d g f 𝔈
      by (intro cat-cf-parallel-2-cat-equalizer prems' f)
    then have ↑↑→↑↑_CF 𝔈 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL v d g f :
      ↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL ↦↦_C.tmα 𝔈
      by (intro cf-parallel-2.cf-parallel-2-the-cf-parallel-2-is-tm-functor)
    from cat-small-complete[OF this] obtain ε' u
      where ε' :
        u <_CF.lim ↑↑→↑↑_CF 𝔈 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL v d g f :
        ↑↑_C 𝔞_PL2 𝔟_PL2 𝔤_PL 𝔣_PL ↦↦_Cα 𝔈
      by clarsimp
    from is-cat-equalizer-2I[OF this prems' f] interpret ε':
      is-cat-equalizer-2 α v d g f 𝔈 u ε'.
    note ε'-is-monic-arr = is-cat-equalizer-2.cat-eq-2-is-monic-arr[
      OF ε'.is-cat-equalizer-2-axioms
```

```
    ]
  note ε′-is-monic-arrD = is-monic-arrD[OF ε′-is-monic-arr]
  then have u ∈∘ ℭ(|Obj|) by auto
  from wd[OF this] obtain s where s: s : w ↦ℭ u by clarsimp
  from s ε′-is-monic-arrD(1) ε-is-monic-arrD(1) have εε′s:
    ε(|NTMap|)(|𝔞PL ?ww|) ∘Aℭ ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ s : w ↦ℭ w
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from s ε′-is-monic-arrD(1) ε-is-monic-arrD(1) have ε′sε:
    ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ s ∘Aℭ ε(|NTMap|)(|𝔞PL ?ww|) : v ↦ℭ v
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from ε-is-monic-arrD(1) ε′-is-monic-arrD(1) s have
    ε(|NTMap|)(|𝔞PL ?ww|) ∘Aℭ (ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ s ∘Aℭ ε(|NTMap|)(|𝔞PL ?ww|)) =
      ε(|NTMap|)(|𝔞PL ?ww|) ∘Aℭ ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ s ∘Aℭ ε(|NTMap|)(|𝔞PL ?ww|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  also from
    ε.cat-eq-Comp-eq
      [
        unfolded in-Hom-iff, OF cat-CId-is-arr[OF π.cat-cone-obj] εε′s,
        symmetric
      ]
    εε′s π.cat-cone-obj ε-is-monic-arr(1)
  have ... = ℭ(|CId|)(|w|) ∘Aℭ ε(|NTMap|)(|𝔞PL ?ww|)
    by (cs-prems cs-shallow cs-simp: vid-on-atI cs-intro: cat-cs-intros)
  also from ε.cf-parallel-𝔞′ ε-is-monic-arrD(1) have
    ... = ε(|NTMap|)(|𝔞PL ?ww|) ∘Aℭ ℭ(|CId|)(|v|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  finally have
    ε(|NTMap|)(|𝔞PL ?ww|) ∘Aℭ (ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ s ∘Aℭ ε(|NTMap|)(|𝔞PL ?ww|)) =
      ε(|NTMap|)(|𝔞PL ?ww|) ∘Aℭ ℭ(|CId|)(|v|).
  from
    is-monic-arrD(2)[OF ε-is-monic-arr ε′sε CId-v this]
    ε′-is-monic-arrD(1) s ε-is-monic-arrD(1)
  have ε′sε-is-CId:
    ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ (s ∘Aℭ ε(|NTMap|)(|𝔞PL ?ww|)) = ℭ(|CId|)(|v|)
    by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  have ε′-is-iso-arr: ε′(|NTMap|)(|𝔞PL2|) : u ↦iso ℭ v
    by
    (
      intro
        cat-is-iso-arr-if-is-monic-arr-is-right-inverse
        ε′-is-monic-arr,
      rule is-right-inverseI[OF - ε′-is-monic-arrD(1) ε′sε-is-CId]
    )
    (
      use s ε-is-monic-arrD(1) in
        ‹cs-concl cs-shallow cs-intro: cat-cs-intros›
    )
  from ε′.cat-eq-2-Comp-eq(1) have
    g ∘Aℭ ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ (ε′(|NTMap|)(|𝔞PL2|))⁻¹Cℭ =
      f ∘Aℭ ε′(|NTMap|)(|𝔞PL2|) ∘Aℭ (ε′(|NTMap|)(|𝔞PL2|))⁻¹Cℭ
    by simp
  from this f ε′-is-monic-arrD(1) ε′-is-iso-arr prems′ show g = f
    by
    (
      cs-prems cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-arrow-cs-intros
    )
qed
```

qed

qed

lemma (**in** *cat-small-cocomplete*) *cat-sc-ex-obj-terminal*:
— See Theorem 1 in Chapter V-6 in [9].
assumes $A \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $A \in_{\circ} Vset\ \alpha$
and $\bigwedge c.\ c \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \exists f\ a.\ a \in_{\circ} A \wedge f : c \mapsto_{\mathfrak{C}} a$
obtains z **where** *obj-terminal* $\mathfrak{C}\ z$
using *that*
by
(
rule cat-small-complete.cat-sc-ex-obj-initial[
OF cat-small-complete-op, *unfolded cat-op-simps*, *OF assms*, *simplified*
]
)

10.2.5 Creation of limits, continuity and completeness

lemma
— See Theorem 2 in Chapter V-4 in [9].
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *cat-small-complete* $\alpha\ \mathfrak{B}$
and $\bigwedge \mathfrak{F}\ \mathfrak{J}.\ \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{A} \Longrightarrow \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\bigwedge \mathfrak{F}\ \mathfrak{J}.\ \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{A} \Longrightarrow$ *cf-creates-limits* $\alpha\ \mathfrak{G}\ \mathfrak{F}$
shows *is-tm-cf-continuous-if-cf-creates-limits*: *is-tm-cf-continuous* $\alpha\ \mathfrak{G}$
and *cat-small-complete-if-cf-creates-limits*: *cat-small-complete* $\alpha\ \mathfrak{A}$
proof–

interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *cat-small-complete* $\alpha\ \mathfrak{B}$ **by** (*rule assms*(*2*))

show *is-tm-cf-continuous* $\alpha\ \mathfrak{G}$
proof(*intro is-tm-cf-continuousI*, *rule assms*)
fix $\mathfrak{F}\ \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{A}$
then interpret $\mathfrak{F}$: *is-tm-functor* $\alpha\ \mathfrak{J}\ \mathfrak{A}\ \mathfrak{F}$ **.**
from *cat-small-completeD*(*2*)[*OF assms*(*2*) *assms*(*3*)[*OF prems*]] **obtain** $\psi\ r$
where ψ: $\psi : r <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by *clarsimp*
show *cf-preserves-limits* $\alpha\ \mathfrak{G}\ \mathfrak{F}$
by
(
rule cf-preserves-limits-if-cf-creates-limits,
rule assms(*1*),
rule $\mathfrak{F}$*.is-functor-axioms*,
rule ψ,
rule assms(*4*)[*OF prems*]
)
qed

show *cat-small-complete* $\alpha\ \mathfrak{A}$
proof(*intro cat-small-completeI* $\mathfrak{G}$*.HomDom.category-axioms*)
fix $\mathfrak{F}\ \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{A}$
then interpret $\mathfrak{F}$: *is-tm-functor* $\alpha\ \mathfrak{J}\ \mathfrak{A}\ \mathfrak{F}$ **.**
from *cat-small-completeD*(*2*)[*OF assms*(*2*) *assms*(*3*)[*OF prems*]] **obtain** $\psi\ r$
where ψ: $\psi : r <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

by *clarsimp*
from *cf-creates-limitsE''*[
OF assms(*4*)[*OF prems*] ψ $\mathfrak{F}$.*is-functor-axioms assms*(*1*)
]
show $\exists u\ r.\ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by *metis*
qed

qed

10.3 Finite-complete and finite-cocomplete category

locale *cat-finite-complete* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
assumes *cat-finite-complete*:
$\bigwedge \mathfrak{F}\ \mathfrak{J}.$ [[*finite-category* α $\mathfrak{J}$; $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$]] $\Longrightarrow$
$\exists u\ r.\ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

locale *cat-finite-cocomplete* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
assumes *cat-finite-cocomplete*:
$\bigwedge \mathfrak{F}\ \mathfrak{J}.$ [[*finite-category* α $\mathfrak{J}$; $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$]] $\Longrightarrow$
$\exists u\ r.\ u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

Rules.

mk-ide rf *cat-finite-complete-def*[*unfolded cat-finite-complete-axioms-def*]
|*intro cat-finite-completeI*|
|*dest cat-finite-completeD*[*dest*]|
|*elim cat-finite-completeE*[*elim*]|

lemma *cat-finite-completeE'*[*elim*]:
assumes *cat-finite-complete* α $\mathfrak{C}$
and *finite-category* α $\mathfrak{J}$
and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains $u\ r$ **where** $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** *auto*

mk-ide rf *cat-finite-cocomplete-def*[*unfolded cat-finite-cocomplete-axioms-def*]
|*intro cat-finite-cocompleteI*|
|*dest cat-finite-cocompleteD*[*dest*]|
|*elim cat-finite-cocompleteE*[*elim*]|

lemma *cat-finite-cocompleteE'*[*elim*]:
assumes *cat-finite-cocomplete* α $\mathfrak{C}$
and *finite-category* α $\mathfrak{J}$
and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains $u\ r$ **where** $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** *auto*

Elementary properties.

sublocale *cat-small-complete* $\subseteq$ *cat-finite-complete*
proof(*intro cat-finite-completeI*)
fix $\mathfrak{F}$ $\mathfrak{J}$ **assume** *prems*: *finite-category* α $\mathfrak{J}$ $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule prems*(*2*))
from *cat-small-complete-axioms* **show** $\exists u\ r.\ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*auto intro*: $\mathfrak{F}$.*cf-is-tm-functor-if-HomDom-finite-category*[*OF prems*(*1*)])
qed (*auto intro*: *cat-cs-intros*)

sublocale *cat-small-cocomplete* $\subseteq$ *cat-finite-cocomplete*
proof(*intro cat-finite-cocompleteI*)

fix $\mathfrak{F}$ $\mathfrak{J}$ **assume** *prems*: *finite-category* α $\mathfrak{J}$ $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule prems*(*2*))
from *cat-small-cocomplete-axioms* **show** $\exists\, u\ r.\ u : \mathfrak{F} >_{CF.colim}\ r : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*auto intro*: $\mathfrak{F}$.*cf-is-tm-functor-if-HomDom-finite-category*[*OF prems*(*1*)])
qed (*auto intro*: *cat-cs-intros*)

11 Comma categories and universal constructions

11.1 Relationship between the universal arrows, initial objects and terminal objects

lemma (in *is-functor*) *universal-arrow-of-if-obj-initial*:
— See Chapter III-1 in [9].
assumes $c \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** *obj-initial* $(c \downarrow_{CF} \mathfrak{F})$ $[0, r, u]_\circ$
shows *universal-arrow-of* $\mathfrak{F}$ c r u
proof(*intro universal-arrow-ofI*)
have *ru*: $[0, r, u]_\circ \in_\circ c \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
and *f-unique*: $C \in_\circ c \downarrow_{CF} \mathfrak{F}(\!|Obj|\!) \Longrightarrow \exists!f.\ f : [0, r, u]_\circ \mapsto_{c \downarrow_{CF} \mathfrak{F}} C$
for C
by (*intro obj-initialD*[*OF assms*(*2*)])+
show *r*: $r \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** *u*: $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
by (*intro cat-obj-cf-comma-ObjD*[*OF ru assms*(*1*)])+
fix r' u' **assume** *prems*: $r' \in_\circ \mathfrak{A}(\!|Obj|\!)$ $u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!)$
from *assms*(*1*) *prems* **have** $r'u'$: $[0, r', u']_\circ \in_\circ c \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
from *f-unique*[*OF* $r'u'$] **obtain** F
where F: $F : [0, r, u]_\circ \mapsto_{c \downarrow_{CF} \mathfrak{F}} [0, r', u']_\circ$
and *F-unique*: $F' : [0, r, u]_\circ \mapsto_{c \downarrow_{CF} \mathfrak{F}} [0, r', u']_\circ \Longrightarrow F' = F$
for F'
by *metis*
from *cat-obj-cf-comma-is-arrE*[*OF F assms*(*1*), *simplified*] **obtain** t
where *F-def*: $F = [[0, r, u]_\circ, [0, r', u']_\circ, [0, t]_\circ]_\circ$
and t: $t : r \mapsto_{\mathfrak{A}} r'$
and [*cat-cs-simps*]: $\mathfrak{F}(\!|ArrMap|\!)(\!|t|\!) \circ_{A\mathfrak{B}} u = u'$
by *metis*
show $\exists!f'.\ f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f'|\!)$
proof(*intro ex1I conjI*; (*elim conjE*)*?*; (*rule t*)*?*)
from t u **show** $u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|t|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
fix t' **assume** *prems'*: $t' : r \mapsto_{\mathfrak{A}} r'$ $u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|t'|\!)$
from *prems'*(*2,1*) u **have** [*symmetric*, *cat-cs-simps*]:
$u' = \mathfrak{F}(\!|ArrMap|\!)(\!|t'|\!) \circ_{A\mathfrak{B}} u$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
define F' **where** $F' = [[0, r, u]_\circ, [0, r', u']_\circ, [0, t']_\circ]_\circ$
from *assms*(*1*) *prems'*(*1*) u *prems*(*2*) **have** F':
$F' : [0, r, u]_\circ \mapsto_{c \downarrow_{CF} \mathfrak{F}} [0, r', u']_\circ$
unfolding *F'-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-comma-cs-intros*)
from *F-unique*[*OF this*] **show** $t' = t$ **unfolding** *F'-def F-def* **by** *simp*
qed
qed

lemma (in *is-functor*) *obj-initial-if-universal-arrow-of*:
— See Chapter III-1 in [9].
assumes *universal-arrow-of* $\mathfrak{F}$ c r u
shows *obj-initial* $(c \downarrow_{CF} \mathfrak{F})$ $[0, r, u]_\circ$
proof–
from *universal-arrow-ofD*[*OF assms*] **have** *r*: $r \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *u*: $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
and *up*: $[\![\ r' \in_\circ \mathfrak{A}(\!|Obj|\!);\ u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|r'|\!)\ ]\!] \Longrightarrow$
$\exists!f'.\ f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = umap\text{-}of\ \mathfrak{F}\ c\ r\ u\ r'(\!|ArrVal|\!)(\!|f'|\!)$
for r' u'
by *auto*
from u **have** *c*: $c \in_\circ \mathfrak{B}(\!|Obj|\!)$ **by** *auto*

```
  show ?thesis
  proof(intro obj-initialI)
    from r u show [0, r, u]ₒ ∈ₒ c ↓CF 𝔉(|Obj|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
    fix B assume prems: B ∈ₒ c ↓CF 𝔉(|Obj|)
    from cat-obj-cf-comma-ObjE[OF prems c] obtain r' u'
      where B-def: B = [0, r', u']ₒ
        and r': r' ∈ₒ 𝔄(|Obj|)
        and u': u' : c ↦𝔅 𝔉(|ObjMap|)(|r'|)
      by auto
    from up[OF r' u'] obtain f
      where f: f : r ↦𝔄 r'
        and u'-def: u' = umap-of 𝔉 c r u r'(|ArrVal|)(|f|)
        and up': ⟦ f' : r ↦𝔄 r'; u' = umap-of 𝔉 c r u r'(|ArrVal|)(|f'|) ⟧ ⟹
          f' = f
      for f'
      by auto
    from u'-def f u have [symmetric, cat-cs-simps]: u' = 𝔉(|ArrMap|)(|f|) ∘A𝔅 u
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    define F where F = [[0, r, u]ₒ, [0, r', u']ₒ, [0, f]ₒ]ₒ
    show ∃!f. f : [0, r, u]ₒ ↦c ↓CF 𝔉 B
      unfolding B-def
    proof(rule ex1I)
      from c u f u' show F : [0, r, u]ₒ ↦c ↓CF 𝔉 [0, r', u']ₒ
        unfolding F-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cs-intro: cat-comma-cs-intros
          )
      fix F' assume prems': F' : [0, r, u]ₒ ↦c ↓CF 𝔉 [0, r', u']ₒ
      from cat-obj-cf-comma-is-arrE[OF prems' c, simplified] obtain f'
        where F'-def: F' = [[0, r, u]ₒ, [0, r', u']ₒ, [0, f']ₒ]ₒ
          and f': f' : r ↦𝔄 r'
          and [cat-cs-simps]: 𝔉(|ArrMap|)(|f'|) ∘A𝔅 u = u'
        by auto
      from f' u have u' = umap-of 𝔉 c r u r'(|ArrVal|)(|f'|)
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      from up'[OF f' this] show F' = F unfolding F'-def F-def by simp
    qed
  qed
qed

lemma (in is-functor) universal-arrow-fo-if-obj-terminal:
  — See Chapter III-1 in [9].
  assumes c ∈ₒ 𝔅(|Obj|) and obj-terminal (𝔉 CF↓ c) [r, 0, u]ₒ
  shows universal-arrow-fo 𝔉 c r u
proof–
  let ?op-𝔉c = ‹op-cat (𝔉 CF↓ c)›
    and ?c-op-𝔉 = ‹c ↓CF (op-cf 𝔉)›
    and ?iso = ‹op-cf-obj-comma 𝔉 c›
  from cat-cf-obj-comma-ObjD[OF obj-terminalD(1)[OF assms(2)] assms(1)]
  have r: r ∈ₒ 𝔄(|Obj|) and u: u : 𝔉(|ObjMap|)(|r|) ↦𝔅 c
    by simp-all
  interpret 𝔉c: is-iso-functor α ?op-𝔉c ?c-op-𝔉 ?iso
    by (rule op-cf-obj-comma-is-iso-functor[OF assms(1)])
  have iso-cocontinuous: is-cf-cocontinuous α ?iso
    by
```

```
    (
      rule is-iso-functor.iso-cf-is-cf-cocontinuous[
        OF 𝔉c.is-iso-functor-axioms
        ]
    )
have iso-preserves: cf-preserves-colimits α ?iso (cf-0 ?op-𝔉c)
  by
    (
      rule is-cf-cocontinuousD
        [
          OF
            iso-cocontinuous
            cf-0-is-functor[OF 𝔉c.HomDom.category-axioms]
            𝔉c.is-functor-axioms
        ]
    )
from category.cat-obj-initial-is-cat-obj-empty-initial[
    OF 𝔉c.HomDom.category-axioms op-cat-obj-initial[THEN iffD2, OF assms(2)]
    ]
interpret ntcf-0-op-𝔉c:
  is-cat-obj-empty-initial α ?op-𝔉c ‹[r, 0, u]∘› ‹ntcf-0 ?op-𝔉c›
  by simp
have cf-0 ?op-𝔉c : cat-0 ↦↦Cα ?op-𝔉c
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
from
  cf-preserves-colimitsD
    [
      OF
        iso-preserves
        ntcf-0-op-𝔉c.is-cat-colimit-axioms
        this
        𝔉c.is-functor-axioms
    ]
  assms(1) r u
have ntcf-0 ?c-op-𝔉 :
  cf-0 ?c-op-𝔉 >CF.colim [0, r, u]∘ : cat-0 ↦↦Cα ?c-op-𝔉
  by
    (
      cs-prems cs-shallow
        cs-simp: cat-cs-simps cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )
then have obj-initial-ru: obj-initial ?c-op-𝔉 [0, r, u]∘
  by
    (
      rule is-cat-obj-empty-initial.cat-oei-obj-initial[
        OF is-cat-obj-empty-initialI
        ]
    )
from assms(1) have c ∈∘ op-cat 𝔅(|Obj|)
  by (cs-concl cs-shallow cs-intro: cat-op-intros)
from
  is-functor.universal-arrow-of-if-obj-initial[
    OF is-functor-op this obj-initial-ru
    ]
have universal-arrow-of (op-cf 𝔉) c r u
  by simp
then show ?thesis unfolding cat-op-simps .
```

qed

lemma (**in** *is-functor*) *obj-terminal-if-universal-arrow-fo*:
— See Chapter III-1 in [9].
assumes *universal-arrow-fo* $\mathfrak{F}$ *c r u*
shows *obj-terminal* ($\mathfrak{F}$ $_{CF}\!\downarrow$ *c*) $[r,\ 0,\ u]_\circ$
proof–
let *?op-*$\mathfrak{F}c$ = ‹*op-cat* ($\mathfrak{F}$ $_{CF}\!\downarrow$ *c*)›
and *?c-op-*$\mathfrak{F}$ = ‹*c* $\downarrow_{CF}$ (*op-cf* $\mathfrak{F}$)›
and *?iso* = ‹*inv-cf* (*op-cf-obj-comma* $\mathfrak{F}$ *c*)›
from *universal-arrow-foD*[*OF assms*] **have** *r*: $r \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *u*: $u : \mathfrak{F}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{\mathfrak{B}} c$
by *auto*
then have *c*: $c \in_\circ \mathfrak{B}(\!|Obj|\!)$ **by** *auto*
from *u* **have** *c-op-*$\mathfrak{F}$: *category* α *?c-op-*$\mathfrak{F}$
by
(
cs-concl **cs-shallow cs-intro**:
cat-cs-intros cat-comma-cs-intros cat-op-intros
)
interpret $\mathfrak{F}c$: *is-iso-functor* α *?op-*$\mathfrak{F}c$ *?c-op-*$\mathfrak{F}$ ‹*op-cf-obj-comma* $\mathfrak{F}$ *c*›
by (*rule op-cf-obj-comma-is-iso-functor*[*OF c*])
interpret *inv-*$\mathfrak{F}c$: *is-iso-functor* α *?c-op-*$\mathfrak{F}$ *?op-*$\mathfrak{F}c$ *?iso*
by (*cs-concl* **cs-shallow cs-intro**: *cf-cs-intros*)
have *iso-cocontinuous*: *is-cf-cocontinuous* α *?iso*
by
(
rule is-iso-functor.iso-cf-is-cf-cocontinuous[
OF inv-$\mathfrak{F}c$*.is-iso-functor-axioms*
]
)
have *iso-preserves*: *cf-preserves-colimits* α *?iso* (*cf-0 ?c-op-*$\mathfrak{F}$)
by
(
rule is-cf-cocontinuousD
[
OF
iso-cocontinuous
cf-0-is-functor[*OF* $\mathfrak{F}c$*.HomCod.category-axioms*]
inv-$\mathfrak{F}c$*.is-functor-axioms*
]
)
from *assms* **have** *universal-arrow-of* (*op-cf* $\mathfrak{F}$) *c r u* **unfolding** *cat-op-simps*.
from *is-cat-obj-empty-initialD*
[
OF category.cat-obj-initial-is-cat-obj-empty-initial
[
OF c-op-$\mathfrak{F}$ *is-functor.obj-initial-if-universal-arrow-of*[
OF is-functor-op this
]
]
]
have *ntcf-0-c-op-*$\mathfrak{F}$: *ntcf-0 ?c-op-*$\mathfrak{F}$:
cf-0 ?c-op-$\mathfrak{F}$ $>_{CF.colim}$ $[0,\ r,\ u]_\circ$: *cat-0* $\mapsto\!\!\mapsto_{C\alpha}$ *?c-op-*$\mathfrak{F}$.
have *cf-0-c-op-*$\mathfrak{F}$: *cf-0 ?c-op-*$\mathfrak{F}$: *cat-0* $\mapsto\!\!\mapsto_{C\alpha}$ *?c-op-*$\mathfrak{F}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from
cf-preserves-colimitsD[

OF iso-preserves ntcf-0-c-op-𝔉 cf-0-c-op-𝔉 inv-𝔉c.is-functor-axioms
]
r u
have *ntcf-0 ?op-𝔉c : cf-0 ?op-𝔉c >$_{CF.colim}$ [r, 0, u]$_\circ$: cat-0 ↦↦$_{C\alpha}$?op-𝔉c*
by
(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps cat-comma-cs-simps 𝔉c.inv-cf-ObjMap-app*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
from
is-cat-obj-empty-initial.cat-oei-obj-initial[
OF is-cat-obj-empty-initialI[*OF this*]
]
show *obj-terminal* (𝔉 $_{CF}$↓ *c*) [*r*, *0*, *u*]$_\circ$
unfolding *op-cat-obj-initial*[*symmetric*].
qed

11.2 A projection for a comma category constructed from a functor and an object creates small limits

See Chapter V-6 in [9].

lemma *cf-obj-cf-comma-proj-creates-limits*:
assumes 𝔊 : 𝔄 ↦↦$_{C\alpha}$ 𝔛
and *is-tm-cf-continuous α 𝔊*
and *x* ∈$_\circ$ 𝔛(⁅*Obj*⁆)
and 𝔉 : 𝔍 ↦↦$_{C.tm\alpha}$ *x* ↓$_{CF}$ 𝔊
shows *cf-creates-limits α* (*x* $_O$∏$_{CF}$ 𝔊) 𝔉
proof(*intro cf-creates-limitsI conjI allI impI*)

interpret 𝔊: *is-functor α 𝔄 𝔛 𝔊* **by** (*rule assms*(*1*))
interpret 𝔉: *is-tm-functor α 𝔍* ‹*x* ↓$_{CF}$ 𝔊› 𝔉 **by** (*rule assms*(*4*))
interpret *x*𝔊: *is-functor α* ‹*x* ↓$_{CF}$ 𝔊› 𝔄 ‹*x* $_O$∏$_{CF}$ 𝔊›
by (*rule 𝔊.cf-obj-cf-comma-proj-is-functor*[*OF assms*(*3*)])

show 𝔉 : 𝔍 ↦↦$_{C\alpha}$ *x* ↓$_{CF}$ 𝔊 **by** (*rule 𝔉.is-functor-axioms*)
show *x* $_O$∏$_{CF}$ 𝔊 : *x* ↓$_{CF}$ 𝔊 ↦↦$_{C\alpha}$ 𝔄 **by** (*rule x*𝔊*.is-functor-axioms*)

define *ψ* :: *V*
where *ψ* =
[
(λ*j*∈$_\circ$𝔍(⁅*Obj*⁆). 𝔉(⁅*ObjMap*⁆)(⁅*j*⁆)(⁅$2_{\mathbb{N}}$⁆)),
cf-const 𝔍 𝔛 *x*,
𝔊 ∘$_{CF}$ (*x* $_O$∏$_{CF}$ 𝔊 ∘$_{CF}$ 𝔉),
𝔍,
𝔛
]$_\circ$

have *ψ-components*:
ψ(⁅*NTMap*⁆) = (λ*j*∈$_\circ$𝔍(⁅*Obj*⁆). 𝔉(⁅*ObjMap*⁆)(⁅*j*⁆)(⁅$2_{\mathbb{N}}$⁆))
ψ(⁅*NTDom*⁆) = *cf-const* 𝔍 𝔛 *x*
ψ(⁅*NTCod*⁆) = 𝔊 ∘$_{CF}$ (*x* $_O$∏$_{CF}$ 𝔊 ∘$_{CF}$ 𝔉)
ψ(⁅*NTDGDom*⁆) = 𝔍
ψ(⁅*NTDGCod*⁆) = 𝔛
unfolding *ψ-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

have *ψ-NTMap-app*: *ψ*(⁅*NTMap*⁆)(⁅*j*⁆) = *f*

```
  if j ∈∘ ℑ(|Obj|) and ℱ(|ObjMap|)(|j|) = [a, b, f]∘ for a b f j
  using that unfolding ψ-components by (simp add: nat-omega-simps)

interpret ψ: is-cat-cone α x ℑ 𝔛 ‹𝔊 ∘CF (x ∘CF.⨅ CF 𝔊 ∘CF ℱ)› ψ
proof(intro is-cat-coneI is-ntcfI')
  show vfsequence ψ unfolding ψ-def by clarsimp
  show vcard ψ = 5ℕ unfolding ψ-def by (simp add: nat-omega-simps)
  show ψ(|NTMap|)(|a|) :
    cf-const ℑ 𝔛 x(|ObjMap|)(|a|) ↦𝔛 (𝔊 ∘CF (x ∘CF.⨅ CF 𝔊 ∘CF ℱ))(|ObjMap|)(|a|)
    if a ∈∘ ℑ(|Obj|) for a
  proof–
    from that have ℱ(|ObjMap|)(|a|) ∈∘ x ↓CF 𝔊(|Obj|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from 𝔊.cat-obj-cf-comma-ObjE[OF this assms(3)] obtain c g
      where ℱa-def: ℱ(|ObjMap|)(|a|) = [0, c, g]∘
        and c: c ∈∘ 𝔄(|Obj|)
        and g: g : x ↦𝔛 𝔊(|ObjMap|)(|c|)
      by auto
    from c g show ?thesis
      using that
      by
        (
          cs-concl
            cs-simp: cat-comma-cs-simps cat-cs-simps ℱa-def ψ-NTMap-app
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed
  show
    ψ(|NTMap|)(|b|) ∘A𝔛 cf-const ℑ 𝔛 x(|ArrMap|)(|f|) =
      (𝔊 ∘CF (x ∘CF.⨅ CF 𝔊 ∘CF ℱ))(|ArrMap|)(|f|) ∘A𝔛 ψ(|NTMap|)(|a|)
    if f : a ↦ℑ b for a b f
  proof–
    from that have ℱf: ℱ(|ArrMap|)(|f|) : ℱ(|ObjMap|)(|a|) ↦x ↓CF 𝔊 ℱ(|ObjMap|)(|b|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from 𝔊.cat-obj-cf-comma-is-arrE[OF this assms(3)] obtain c h c' h' k
      where ℱf-def: ℱ(|ArrMap|)(|f|) = [[0, c, h]∘, [0, c', h']∘, [0, k]∘]∘
        and ℱa-def: ℱ(|ObjMap|)(|a|) = [0, c, h]∘
        and ℱb-def: ℱ(|ObjMap|)(|b|) = [0, c', h']∘
        and k: k : c ↦𝔄 c'
        and h: h : x ↦𝔛 𝔊(|ObjMap|)(|c|)
        and h': h' : x ↦𝔛 𝔊(|ObjMap|)(|c'|)
        and [cat-cs-simps]: 𝔊(|ArrMap|)(|k|) ∘A𝔛 h = h'
      by metis
    from ℱf k h h' that show ?thesis
      unfolding ℱf-def ℱa-def ℱb-def
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps cat-comma-cs-simps
              ℱf-def ℱa-def ℱb-def ψ-NTMap-app
            cs-intro: cat-cs-intros
        )
  qed
qed (auto simp: assms(3) ψ-components intro: cat-cs-intros)

fix τ b assume prems: τ : b <CF.lim x ∘CF.⨅ CF 𝔊 ∘CF ℱ : ℑ ↦↦Cα 𝔄
interpret τ: is-cat-limit α ℑ 𝔄 ‹x ∘CF.⨅ CF 𝔊 ∘CF ℱ› b τ by (rule prems)
```

note $x\mathfrak{G}\text{-}\mathfrak{F}$ = *cf-comp-cf-obj-cf-comma-proj-is-tm-functor*[*OF assms*(*1,4,3*)]
have *cf-preserves-limits* α $\mathfrak{G}$ (x $_{O}\bigsqcap_{CF}$ $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$)
  by (*rule is-tm-cf-continuousD* [*OF assms*(*2*) $x\mathfrak{G}\text{-}\mathfrak{F}$ *assms*(*1*)])
then have $\mathfrak{G}\tau$: $\mathfrak{G}$ $\circ_{CF-NTCF}$ τ :
  $\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$ $<_{CF.lim}$ $\mathfrak{G}$ $\circ_{CF}$ (x $_{O}\bigsqcap_{CF}$ $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$) : $\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{X}$
  by
    (
      rule cf-preserves-limitsD[
        OF - prems(*1*) *is-tm-functorD*(*1*)[*OF* $x\mathfrak{G}\text{-}\mathfrak{F}$] *assms*(*1*)
        ]
)

from *is-cat-limit.cat-lim-unique-cone'*[*OF* $\mathfrak{G}\tau$ ψ.*is-cat-cone-axioms*] **obtain** f
  where f: $f : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$
    and ψ-f: $\bigwedge j.\ j \in_{\circ} \mathfrak{J}(\!|Obj|\!) \Longrightarrow$
      $\psi(\!|NTMap|\!)(\!|j|\!) = (\mathfrak{G} \circ_{CF-NTCF} \tau)(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{X}} f$
    and *f-unique*:
      [[
        $f' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$;
        $\bigwedge j.\ j \in_{\circ} \mathfrak{J}(\!|Obj|\!) \Longrightarrow \psi(\!|NTMap|\!)(\!|j|\!) = (\mathfrak{G} \circ_{CF-NTCF} \tau)(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{X}} f'$
       ]] $\Longrightarrow f' = f$
  for f'
  by *metis*

define σ :: V
  where σ =
    [
      (
        $\lambda j \in_{\circ} \mathfrak{J}(\!|Obj|\!)$.
          [
            [0, b, f]$_{\circ}$,
            [0, (x $_{O}\bigsqcap_{CF}$ $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$)$(\!|ObjMap|\!)(\!|j|\!)$, $\psi(\!|NTMap|\!)(\!|j|\!)$]$_{\circ}$,
            [0, $\tau(\!|NTMap|\!)(\!|j|\!)$]$_{\circ}$
          ]$_{\circ}$
      ),
      cf-const $\mathfrak{J}$ (x $\downarrow_{CF}$ $\mathfrak{G}$) [0, b, f]$_{\circ}$,
      $\mathfrak{F}$,
      $\mathfrak{J}$,
      x $\downarrow_{CF}$ $\mathfrak{G}$
    ]$_{\circ}$

have σ-*components*: $\sigma(\!|NTMap|\!)$ =
    (
      $\lambda j \in_{\circ} \mathfrak{J}(\!|Obj|\!)$.
        [
          [0, b, f]$_{\circ}$,
          [0, (x $_{O}\bigsqcap_{CF}$ $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$)$(\!|ObjMap|\!)(\!|j|\!)$, $\psi(\!|NTMap|\!)(\!|j|\!)$]$_{\circ}$,
          [0, $\tau(\!|NTMap|\!)(\!|j|\!)$]$_{\circ}$
        ]$_{\circ}$
)
  and [*cat-cs-simps*]: $\sigma(\!|NTDom|\!)$ = *cf-const* $\mathfrak{J}$ (x $\downarrow_{CF}$ $\mathfrak{G}$) [0, b, f]$_{\circ}$
  and [*cat-cs-simps*]: $\sigma(\!|NTCod|\!)$ = $\mathfrak{F}$
  and [*cat-cs-simps*]: $\sigma(\!|NTDGDom|\!)$ = $\mathfrak{J}$
  and [*cat-cs-simps*]: $\sigma(\!|NTDGCod|\!)$ = x $\downarrow_{CF}$ $\mathfrak{G}$
  unfolding σ-*def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

have σ-*NTMap-app*: $\sigma(\!|NTMap|\!)(\!|j|\!)$ =

```
  [
    [0, b, f]_∘,
    [0, (x _O∏_CF 𝔊 ∘_CF 𝔉)(|ObjMap|)(|j|), ψ(|NTMap|)(|j|)]_∘,
    [0, τ(|NTMap|)(|j|)]_∘
  ]_∘
  if j ∈_∘ 𝔍(|Obj|) for j
  using that unfolding σ-components by simp

interpret σ: is-cat-cone α ‹[0, b, f]_∘› 𝔍 ‹x ↓_CF 𝔊› 𝔉 σ
proof(intro is-cat-coneI is-ntcfI')
  show vfsequence σ unfolding σ-def by auto
  show vcard σ = 5_ℕ unfolding σ-def by (simp add: nat-omega-simps)
  from f show cf-const 𝔍 (x ↓_CF 𝔊) [0, b, f]_∘ : 𝔍 ↦↦_Cα x ↓_CF 𝔊
    by
      (
        cs-concl cs-intro:
          cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
      )
  show vsv (σ(|NTMap|)) unfolding σ-components by auto
  show 𝒟_∘ (σ(|NTMap|)) = 𝔍(|Obj|) unfolding σ-components by auto
  from assms(3) show σ(|NTMap|)(|a|) :
    cf-const 𝔍 (x ↓_CF 𝔊) [0, b, f]_∘(|ObjMap|)(|a|) ↦_x ↓_CF 𝔊 𝔉(|ObjMap|)(|a|)
    if a ∈_∘ 𝔍(|Obj|) for a
  proof–
    from that have 𝔉a: 𝔉(|ObjMap|)(|a|) ∈_∘ x ↓_CF 𝔊(|Obj|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from 𝔊.cat-obj-cf-comma-ObjE[OF this assms(3)] obtain c g
      where 𝔉a-def: 𝔉(|ObjMap|)(|a|) = [0, c, g]_∘
        and c: c ∈_∘ 𝔄(|Obj|)
        and g: g : x ↦_𝔛 𝔊(|ObjMap|)(|c|)
      by auto
    from ψ-f[OF that] that c f g 𝔉a have [symmetric, cat-cs-simps]:
      g = 𝔊(|ArrMap|)(|τ(|NTMap|)(|a|)|) ∘_A𝔛 f
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps ψ-NTMap-app 𝔉a-def cs-intro: cat-cs-intros
        )
    from that c f g 𝔉a show ?thesis
      unfolding 𝔉a-def
      by
        (
          cs-concl
            cs-simp:
              cat-comma-cs-simps cat-cs-simps
              ψ-NTMap-app σ-NTMap-app 𝔉a-def
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed
  show
    σ(|NTMap|)(|d|) ∘_Ax ↓_CF 𝔊 cf-const 𝔍 (x ↓_CF 𝔊) [0, b, f]_∘(|ArrMap|)(|g|) =
      𝔉(|ArrMap|)(|g|) ∘_Ax ↓_CF 𝔊 σ(|NTMap|)(|c|)
    if g : c ↦_𝔍 d for c d g
  proof–
    from that have 𝔉g: 𝔉(|ArrMap|)(|g|) : 𝔉(|ObjMap|)(|c|) ↦_x ↓_CF 𝔊 𝔉(|ObjMap|)(|d|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from 𝔊.cat-obj-cf-comma-is-arrE[OF this assms(3)] obtain e h e' h' k
      where 𝔉g-def: 𝔉(|ArrMap|)(|g|) = [[0, e, h]_∘, [0, e', h']_∘, [0, k]_∘]_∘
```

```
      and 𝔉c-def: 𝔉(⁅ObjMap⁆)(⁅c⁆) = [0, e, h]∘
      and 𝔉d-def: 𝔉(⁅ObjMap⁆)(⁅d⁆) = [0, e′, h′]∘
      and k: k : e ↦𝔄 e′
      and h: h : x ↦𝔛 𝔊(⁅ObjMap⁆)(⁅e⁆)
      and h′: h′ : x ↦𝔛 𝔊(⁅ObjMap⁆)(⁅e′⁆)
      and [cat-cs-simps]: 𝔊(⁅ArrMap⁆)(⁅k⁆) ∘A𝔛 h = h′
    by metis
  from that have c ∈∘ ℑ(⁅Obj⁆) by auto
  from ψ-f[OF this] that k f h have [symmetric, cat-cs-simps]:
    h = 𝔊(⁅ArrMap⁆)(⁅τ(⁅NTMap⁆)(⁅c⁆)⁆) ∘A𝔛 f
    by
      (
        cs-prems
          cs-simp: cat-cs-simps ψ-NTMap-app 𝔉c-def cs-intro: cat-cs-intros
      )
  from that have d ∈∘ ℑ(⁅Obj⁆)by auto
  from ψ-f[OF this] that k f h′ have [symmetric, cat-cs-simps]:
    h′ = 𝔊(⁅ArrMap⁆)(⁅τ(⁅NTMap⁆)(⁅d⁆)⁆) ∘A𝔛 f
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps ψ-NTMap-app 𝔉d-def cs-intro: cat-cs-intros
      )
  note τ.cat-cone-Comp-commute[cat-cs-simps del]
  from τ.ntcf-Comp-commute[OF that] that assms(3) k h h′
  have [symmetric, cat-cs-simps]: τ(⁅NTMap⁆)(⁅d⁆) = k ∘A𝔄 τ(⁅NTMap⁆)(⁅c⁆)
    by
      (
        cs-prems
          cs-simp: cat-comma-cs-simps cat-cs-simps 𝔉g-def 𝔉c-def 𝔉d-def
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  from that f 𝔉g k h h′ show ?thesis
    unfolding 𝔉g-def 𝔉c-def 𝔉d-def
    by
      (
        cs-concl
          cs-simp:
            cat-comma-cs-simps cat-cs-simps
            ψ-NTMap-app σ-NTMap-app 𝔉g-def 𝔉c-def 𝔉d-def
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
 qed
qed
 (
   use f in
     ‹
       cs-concl cs-shallow
         cs-intro: cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
         cs-simp: cat-cs-simps
     ›
 )+

have τσ: τ = x O⨅CF 𝔊 ∘CF−NTCF σ
proof(rule ntcf-eqI)
  show τ : cf-const ℑ 𝔄 b ↦CF x O⨅CF 𝔊 ∘CF 𝔉 : ℑ ↦↦Cα 𝔄
    by (rule τ.is-ntcf-axioms)
  from f show x O⨅CF 𝔊 ∘CF−NTCF σ :
```

```
    cf-const 𝔍 𝔄 b ↦_CF x _O⨅_CF 𝔊 ∘_CF 𝔉 : 𝔍 ↦↦_Cα 𝔄
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
      )
  have dom-lhs: 𝒟∘ (τ(|NTMap|)) = 𝔍(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have dom-rhs: 𝒟∘ ((x _O⨅_CF 𝔊 ∘_CF-NTCF σ)(|NTMap|)) = 𝔍(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show τ(|NTMap|) = (x _O⨅_CF 𝔊 ∘_CF-NTCF σ)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume prems': a ∈∘ 𝔍(|Obj|)
    then have 𝔉(|ObjMap|)(|a|) ∈∘ x ↓_CF 𝔊(|Obj|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from 𝔊.cat-obj-cf-comma-ObjE[OF this assms(3)] obtain c g
      where 𝔉a-def: 𝔉(|ObjMap|)(|a|) = [0, c, g]∘
        and c: c ∈∘ 𝔄(|Obj|)
        and g: g : x ↦_𝔛 𝔊(|ObjMap|)(|c|)
      by auto
    from ψ-f[OF prems'] prems' f g have [symmetric, cat-cs-simps]:
      g = 𝔊(|ArrMap|)(|τ(|NTMap|)(|a|)|) ∘_A𝔛 f
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps ψ-NTMap-app 𝔉a-def cs-intro: cat-cs-intros
        )
    with assms(3) prems' c g f show
      τ(|NTMap|)(|a|) = (x _O⨅_CF 𝔊 ∘_CF-NTCF σ)(|NTMap|)(|a|)
      by
        (
          cs-concl cs-shallow
            cs-simp:
              cat-comma-cs-simps cat-cs-simps
              𝔉a-def ψ-NTMap-app σ-NTMap-app
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed (simp-all add: τ.ntcf-NTMap-vsv cat-cs-intros)
qed simp-all

from f have b-def: b = x _O⨅_CF 𝔊(|ObjMap|)(|0, b, f|)•
  by (cs-concl cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros)

show σa-unique: ∃!σa. ∃σ a.
  σa = ⟨σ, a⟩ ∧
  σ : a <_CF.cone 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊 ∧
  τ = x _O⨅_CF 𝔊 ∘_CF-NTCF σ ∧ b = x _O⨅_CF 𝔊(|ObjMap|)(|a|)
proof
  (
    intro ex1I[where a=‹⟨σ, [0, b, f]∘⟩›] exI conjI, simp only:;
    (elim exE conjE)?
  )

  show σ : [0, b, f]∘ <_CF.cone 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
    by (rule σ.is-cat-cone-axioms)
  show τ = x _O⨅_CF 𝔊 ∘_CF-NTCF σ by (rule τσ)
  show b = x _O⨅_CF 𝔊(|ObjMap|) (|0, b, f|)• by (rule b-def)
```

fix $\sigma a\ \sigma'\ a$ **assume** *prems'*:
$\sigma a = \langle \sigma', a \rangle$
$\sigma' : a <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
$\tau = x\ {}_{O}\!\sqcap_{CF}\ \mathfrak{G} \circ_{CF-NTCF} \sigma'$
$b = x\ {}_{O}\!\sqcap_{CF}\ \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$
interpret σ': *is-cat-cone* $\alpha\ a\ \mathfrak{J}\ \langle x \downarrow_{CF} \mathfrak{G} \rangle\ \mathfrak{F}\ \sigma'$ **by** (*rule prems'*(*2*))

from $\mathfrak{G}$.*cat-obj-cf-comma-ObjE*[*OF* σ'.*cat-cone-obj assms*(*3*)] **obtain** $c\ g$
where *a-def''*: $a = [0, c, g]_\circ$
and *c'*: $c \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *g'*: $g : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|c|\!)$
by *auto*
from *prems'*(*4*) *c' g' assms*(*3*) **have** *bc*: $b = c$
by
(
cs-prems **cs-shallow**
cs-simp: *cat-comma-cs-simps a-def''* **cs-intro**: *cat-comma-cs-intros*
)
with *a-def'' c' g'* **have** *a-def'*: $a = [0, b, g]_\circ$
and *b*: $b \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *g*: $g : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$
by *auto*

from *prems'*(*3*) **have** τ-*eq-x*$\mathfrak{G}$-σ':
$\tau(\!|NTMap|\!)(\!|j|\!) = (x\ {}_{O}\!\sqcap_{CF}\ \mathfrak{G} \circ_{CF-NTCF} \sigma')(\!|NTMap|\!)(\!|j|\!)$ **for** j
by *simp*

have $\psi(\!|NTMap|\!)(\!|j|\!) = (\mathfrak{G} \circ_{CF-NTCF} \tau)(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{X}} g$
if $j \in_\circ \mathfrak{J}(\!|Obj|\!)$ **for** j
proof–
from *that* **have** σ'-*j*: $\sigma'(\!|NTMap|\!)(\!|j|\!) : [0, b, g]_\circ \mapsto_{x \downarrow_{CF}} \mathfrak{G}\ \mathfrak{F}(\!|ObjMap|\!)(\!|j|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps a-def'*[*symmetric*] **cs-intro**: *cat-cs-intros*
)
from $\mathfrak{G}$.*cat-obj-cf-comma-is-arrE*[*OF this*] **obtain** $e\ h\ k$
where σ'-*j-def*: $\sigma'(\!|NTMap|\!)(\!|j|\!) = [[0, b, g]_\circ, [0, e, h]_\circ, [0, k]_\circ]_\circ$
and $\mathfrak{F}$*j-def*: $\mathfrak{F}(\!|ObjMap|\!)(\!|j|\!) = [0, e, h]_\circ$
and *k*: $k : b \mapsto_{\mathfrak{A}} e$
and *h*: $h : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|e|\!)$
and [*cat-cs-simps*]: $\mathfrak{G}(\!|ArrMap|\!)(\!|k|\!) \circ_{A\mathfrak{X}} g = h$
by (*metis* $\mathfrak{G}$.*cat-obj-cf-comma-is-arrD*(*4*,*7*) σ'-*j assms*(*3*))
from *that* σ'-*j* **show** *?thesis*
unfolding σ'-*j-def*
by
(
cs-concl **cs-shallow**
cs-simp:
cat-cs-simps cat-comma-cs-simps
σ'-*j-def* ψ-*NTMap-app* $\mathfrak{F}$*j-def prems'*(*3*)
cs-intro: *cat-cs-intros*
)
qed
from *f-unique*[*OF g this*] **have** *gf*: $g = f$.
with *a-def'* **have** *a-def*: $a = [0, b, f]_\circ$ **by** *simp*

```
  have σσ': σ = σ'
  proof(rule ntcf-eqI)
    show σ : cf-const 𝔍 (x ↓_CF 𝔊) [0, b, f]_∘ ↦_CF 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    then have dom-lhs: 𝒟_∘ (σ(|NTMap|)) = 𝔍(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    show σ' : cf-const 𝔍 (x ↓_CF 𝔊) [0, b, f]_∘ ↦_CF 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
      by (cs-concl cs-shallow cs-simp: a-def cs-intro: cat-cs-intros)
    then have dom-rhs: 𝒟_∘ (σ'(|NTMap|)) = 𝔍(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    show σ(|NTMap|) = σ'(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix j assume prems'': j ∈_∘ 𝔍(|Obj|)
      then have σ'-j: σ'(|NTMap|)(|j|) : [0, b, f]_∘ ↦_x ↓_CF 𝔊 𝔉(|ObjMap|)(|j|)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps a-def'[symmetric] gf[symmetric]
              cs-intro: cat-cs-intros
          )
      from 𝔊.cat-obj-cf-comma-is-arrE[OF this] obtain e h k
        where σ'-j-def: σ'(|NTMap|)(|j|) = [[0, b, f]_∘, [0, e, h]_∘, [0, k]_∘]_∘
          and 𝔉j-def: 𝔉(|ObjMap|)(|j|) = [0, e, h]_∘
          and k: k : b ↦_𝔄 e
          and h: h : x ↦_𝔛 𝔊(|ObjMap|)(|e|)
          and [cat-cs-simps]: 𝔊(|ArrMap|)(|k|) ∘_A𝔛 f = h
        by (metis 𝔊.cat-obj-cf-comma-is-arrD(4,7) σ'-j assms(3))
      from prems'' k h assms(3) f h show σ(|NTMap|)(|j|) = σ'(|NTMap|)(|j|)
        by
          (
            cs-concl cs-shallow
              cs-simp:
                cat-cs-simps cat-comma-cs-simps
                τ-eq-x𝔊-σ' ψ-NTMap-app σ-NTMap-app 𝔉j-def σ'-j-def
              cs-intro: cat-cs-intros cat-comma-cs-intros
          )
    qed (cs-concl cs-shallow cs-intro: V-cs-intros)
  qed simp-all
  show σa = ⟨σ, [[]_∘, b, f]_∘⟩ unfolding prems'(1) σσ' a-def by simp
qed

show σ' : a' <_CF.lim 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
  if σ' : a' <_CF.cone 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
    and τ = x _O⨅_CF 𝔊 ∘_CF-NTCF σ'
    and b = x _O⨅_CF 𝔊(|ObjMap|)(|a'|)
  for σ' a'
proof(rule is-cat-limitI)

  interpret σ': is-cat-cone α a' 𝔍 ‹x ↓_CF 𝔊› 𝔉 σ' by (rule that(1))

  from σ.is-cat-cone-axioms τσ b-def that σa-unique have
    ⟨σ', a'⟩ = ⟨σ, [0, b, f]_∘⟩
    by metis
  then have σ'-def: σ' = σ and a'-def: a' = [0, b, f]_∘ by simp-all

  show σ' : a' <_CF.cone 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
    by (rule σ'.is-cat-cone-axioms)
```

fix σ'' a'' **assume** *prems*: $\sigma'' : a'' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
then interpret σ'': *is-cat-cone* α a'' $\mathfrak{J}$ $\langle x \downarrow_{CF} \mathfrak{G}\rangle$ $\mathfrak{F}$ σ'' **.**
from $\mathfrak{G}$.*cat-obj-cf-comma-ObjE*[*OF* σ''.*cat-cone-obj assms*(*3*)] **obtain** b' f'
  where a''-*def*: $a'' = [0,\ b',\ f']_\circ$
    and b': $b' \in_\circ \mathfrak{A}(\!|Obj|\!)$
    and f': $f' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|b'|\!)$
  by *auto*
from b' f' **have** $x\mathfrak{G}$-A': $x\ {}_O\!\bigcap_{CF}\ \mathfrak{G}(\!|ObjMap|\!)(\!|a''|\!) = b'$
  unfolding a''-*def*
  by
    (
      cs-concl **cs-shallow**
        cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)

from τ.*cat-lim-unique-cone'*[
    OF cf-ntcf-comp-cf-cat-cone[*OF prems* $x\mathfrak{G}$.*is-functor-axioms*],
    unfolded $x\mathfrak{G}$-A'
    ]
obtain h **where** h: $h : b' \mapsto_{\mathfrak{A}} b$
  and $x\mathfrak{G}$-σ''-*app*: $\bigwedge j.\ j \in_\circ \mathfrak{J}(\!|Obj|\!) \Longrightarrow$
    $(x\ {}_O\!\bigcap_{CF}\ \mathfrak{G} \circ_{CF-NTCF} \sigma'')(\!|NTMap|\!)(\!|j|\!) = \tau(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{A}} h$
  and *h-unique*:
    [[
      $h' : b' \mapsto_{\mathfrak{A}} b$;
      $\bigwedge j.\ j \in_\circ \mathfrak{J}(\!|Obj|\!) \Longrightarrow$
        $(x\ {}_O\!\bigcap_{CF}\ \mathfrak{G} \circ_{CF-NTCF} \sigma'')(\!|NTMap|\!)(\!|j|\!) = \tau(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{A}} h'$
    ]] $\Longrightarrow h' = h$
  for h'
  by *metis*

define F **where** $F = [a'',\ a',\ [0,\ h]_\circ]_\circ$

show $\exists! F'$.
  $F' : a'' \mapsto_{x \downarrow_{CF} \mathfrak{G}} a' \wedge \sigma'' = \sigma' \bullet_{NTCF} \mathit{ntcf\text{-}const}\ \mathfrak{J}\ (x \downarrow_{CF} \mathfrak{G})\ F'$
  unfolding a''-*def* a'-*def* σ'-*def*
proof(*intro ex1I conjI*; (*elim conjE*)*?*)
  from f' h **have** $\mathfrak{G}h$-f': $\mathfrak{G}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{X}} f' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$
    by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
  have $\psi(\!|NTMap|\!)(\!|j|\!) = (\mathfrak{G} \circ_{CF-NTCF} \tau)(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{X}} (\mathfrak{G}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{X}} f')$
    if $j \in_\circ \mathfrak{J}(\!|Obj|\!)$ **for** j
  proof–
    from *that* **have** σ''-j:
      $\sigma''(\!|NTMap|\!)(\!|j|\!) : [0,\ b',\ f']_\circ \mapsto_{x \downarrow_{CF} \mathfrak{G}} \mathfrak{F}(\!|ObjMap|\!)(\!|j|\!)$
      by
        (
          cs-concl **cs-shallow**
            cs-simp: *cat-cs-simps* a''-*def*[*symmetric*]
            cs-intro: *cat-cs-intros*
)
    from $\mathfrak{G}$.*cat-obj-cf-comma-is-arrE*[*OF this*] **obtain** e h' k
      where σ''-j-*def*: $\sigma''(\!|NTMap|\!)(\!|j|\!) = [[0,\ b',\ f']_\circ,\ [0,\ e,\ h']_\circ,\ [0,\ k]_\circ]_\circ$
        and $\mathfrak{F}j$-*def*: $\mathfrak{F}(\!|ObjMap|\!)(\!|j|\!) = [0,\ e,\ h']_\circ$
        and k: $k : b' \mapsto_{\mathfrak{A}} e$
        and [*cat-cs-simps*]: $\mathfrak{G}(\!|ArrMap|\!)(\!|k|\!) \circ_{A\mathfrak{X}} f' = h'$
      by (*metis* $\mathfrak{G}$.*cat-obj-cf-comma-is-arrD*(*4*,*7*) σ''-j *assms*(*3*))
    from *that* σ''-j **have** ψ-*NTMap-j*: $\psi(\!|NTMap|\!)(\!|j|\!) =$
      $(\mathfrak{G} \circ_{CF-NTCF} (x\ {}_O\!\bigcap_{CF}\ \mathfrak{G} \circ_{CF-NTCF} \sigma''))(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{X}} f'$

```
    unfolding σ''-j-def 𝔉j-def
    by
      (
        cs-concl cs-shallow
          cs-simp:
            cat-cs-simps cat-comma-cs-simps σ''-j-def 𝔉j-def ψ-NTMap-app
          cs-intro: cat-cs-intros
      )+
  from that h f' show ?thesis
    unfolding ψ-NTMap-j
    by
      (
        cs-concl cs-shallow
          cs-simp:
            cat-cs-simps
            is-ntcf.cf-ntcf-comp-NTMap-app
            x𝔊-σ''-app[OF that]
          cs-intro: cat-cs-intros
      )
qed

from f-unique[OF 𝔊h-f' this] have [cat-cs-simps]:
  𝔊(‖ArrMap‖)(‖h‖) ∘_A𝔛 f' = f.

from assms(3) h f' f show F: F : [0, b', f']∘ ↦_x ↓_CF 𝔊 [0, b, f]∘
  unfolding F-def a''-def a'-def
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-comma-cs-intros
    )

show σ'' = σ ·_NTCF ntcf-const 𝔍 (x ↓_CF 𝔊) F
proof(rule ntcf-eqI)
  show σ'' : cf-const 𝔍 (x ↓_CF 𝔊) a'' ↦_CF 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
    by (rule σ''.is-ntcf-axioms)
  then have dom-lhs: 𝒟∘ (σ''(‖NTMap‖)) = 𝔍(‖Obj‖)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from F show σ ·_NTCF ntcf-const 𝔍 (x ↓_CF 𝔊) F :
    cf-const 𝔍 (x ↓_CF 𝔊) a'' ↦_CF 𝔉 : 𝔍 ↦↦_Cα x ↓_CF 𝔊
    unfolding a''-def by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  then have dom-rhs:
    𝒟∘ ((σ ·_NTCF ntcf-const 𝔍 (x ↓_CF 𝔊) F)(‖NTMap‖)) = 𝔍(‖Obj‖)
    by (cs-concl cs-simp: cat-cs-simps)
  show σ''(‖NTMap‖) = (σ ·_NTCF ntcf-const 𝔍 (x ↓_CF 𝔊) F)(‖NTMap‖)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix j assume prems': j ∈∘ 𝔍(‖Obj‖)
    then have σ''-j:
      σ''(‖NTMap‖)(‖j‖) : [0, b', f']∘ ↦_x ↓_CF 𝔊 𝔉(‖ObjMap‖)(‖j‖)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps a''-def[symmetric]
            cs-intro: cat-cs-intros
        )
    from 𝔊.cat-obj-cf-comma-is-arrE[OF this] obtain e h' k
      where σ''-j-def: σ''(‖NTMap‖)(‖j‖) = [[0, b', f']∘, [0, e, h']∘, [0, k]∘]∘
        and 𝔉j-def: 𝔉(‖ObjMap‖)(‖j‖) = [0, e, h']∘
```

```
          and k: k : b' ↦_𝔄 e
          and h': h' : x ↦_𝔛 𝔊(|ObjMap|)(|e|)
          and [cat-cs-simps]: 𝔊(|ArrMap|)(|k|) ∘_A𝔛 f' = h'
        by (metis 𝔊.cat-obj-cf-comma-is-arrD(4,7) σ''-j assms(3))
      from assms(3) prems' F k h' h f f' show
       σ''(|NTMap|)(|j|) = (σ ·_NTCF ntcf-const 𝔍 (x ↓_CF 𝔊) F)(|NTMap|)(|j|)
       by
        (
          cs-concl
            cs-simp:
              cat-cs-simps cat-comma-cs-simps
              σ''-j-def x𝔊-σ''-app[OF prems', symmetric]
              σ-NTMap-app F-def 𝔉j-def a''-def a'-def
            cs-intro: cat-cs-intros cat-comma-cs-intros
            cs-simp: ψ-f ψ-NTMap-app
        )
    qed (cs-concl cs-intro: V-cs-intros cat-cs-intros)+
  qed simp-all
  fix F' assume prems':
    F' : [0, b', f']∘ ↦_x ↓_CF 𝔊 [0, b, f]∘
    σ'' = σ ·_NTCF ntcf-const 𝔍 (x ↓_CF 𝔊) F'
  from 𝔊.cat-obj-cf-comma-is-arrE[OF prems'(1) assms(3), simplified]
  obtain k
    where F'-def: F' = [[0, b', f']∘, [0, b, f]∘, [0, k]∘]∘
      and k: k : b' ↦_𝔄 b
      and [cat-cs-simps]: 𝔊(|ArrMap|)(|k|) ∘_A𝔛 f' = f
    by metis
  have k = h
  proof(rule h-unique[OF k])
    fix j assume prems'': j ∈∘ 𝔍(|Obj|)
    then have 𝔉(|ObjMap|)(|j|) ∈∘ x ↓_CF 𝔊(|Obj|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from 𝔊.cat-obj-cf-comma-ObjE[OF this assms(3)] obtain c g
      where 𝔉j-def: 𝔉(|ObjMap|)(|j|) = [0, c, g]∘
        and c: c ∈∘ 𝔄(|Obj|)
        and g: g : x ↦_𝔛 𝔊(|ObjMap|)(|c|)
      by auto
    from prems'' prems'(1) assms(3) c g f show
      (x _O∏_CF 𝔊 ∘_CF-NTCF σ'')(|NTMap|)(|j|) = τ(|NTMap|)(|j|) ∘_A𝔄 k
      unfolding prems'(2) τσ F'-def
      by
        (
          cs-concl
            cs-simp: cat-comma-cs-simps cat-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
            cs-simp: ψ-f σ-NTMap-app 𝔉j-def
        )
  qed
  then show F' = F unfolding F'-def F-def a''-def a'-def by simp
 qed

qed

qed
```

12 Category *Set* and universal constructions

12.1 Discrete functor with tiny maps to the category *Set*

lemma (in $\mathcal{Z}$**)** *tm-cf-discrete-cat-Set-if-VLambda-in-Vset*:
assumes *VLambda I F* $\in_\circ$ *Vset* α
shows *tm-cf-discrete* α *I F* (*cat-Set* α)
proof(*intro tm-cf-discreteI*)
from *assms* **have** *vrange-F-in-Vset*: $\mathcal{R}_\circ$ (*VLambda I F*) $\in_\circ$ *Vset* α
by (*auto intro*: *vrange-in-VsetI*)
show ($\lambda i \in_\circ I$. *cat-Set* $\alpha(\!|CId|\!)(\!|F\ i|\!)$) $\in_\circ$ *Vset* α
proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
from *assms* **show** $\mathcal{D}_\circ$ ($\lambda i \in_\circ I$. *cat-Set* $\alpha(\!|CId|\!)(\!|F\ i|\!)$) $\in_\circ$ *Vset* α
by (*metis vdomain-VLambda vdomain-in-VsetI*)
define *Q* **where**
Q i =
(
if i = 0
then VPow (($\bigcup_\circ i \in_\circ I$. *F i*) $\times_\circ$ ($\bigcup_\circ i \in_\circ I$. *F i*))
else set (*F* ‘ *elts I*)
)
for *i* :: *V*
have $\mathcal{R}_\circ$ ($\lambda i \in_\circ I$. *cat-Set* $\alpha(\!|CId|\!)(\!|F\ i|\!)$) $\subseteq_\circ$ ($\prod_\circ i \in_\circ$ *set* {*0*, $1_{\mathbb{N}}$, $2_{\mathbb{N}}$}. *Q i*)
proof(*intro vsubsetI*, *unfold cat-Set-components*)
fix *y* **assume** *y* $\in_\circ$ $\mathcal{R}_\circ$ ($\lambda i \in_\circ I$. *VLambda* (*Vset* α) *id-Set*$(\!|F\ i|\!)$)
then obtain *i* **where** *i*: *i* $\in_\circ$ *I*
and *y-def*: *y* = *VLambda* (*Vset* α) *id-Set*$(\!|F\ i|\!)$
by *auto*
from *i* **have** *F i* $\in_\circ$ $\mathcal{R}_\circ$ (*VLambda I F*) **by** *auto*
with *vrange-F-in-Vset* **have** *F i* $\in_\circ$ *Vset* α **by** *auto*
then have *y-def*: *y* = *id-Set* (*F i*) **unfolding** *y-def* **by** *auto*
show *y* $\in_\circ$ ($\prod_\circ i \in_\circ$ *set* {*0*, $1_{\mathbb{N}}$, $2_{\mathbb{N}}$}. *Q i*)
unfolding *y-def*
proof(*intro vproductI*, *unfold Ball-def*; (*intro allI impI*)*?*)
show $\mathcal{D}_\circ$ (*id-Rel* (*F i*)) = *set* {*0*, $1_{\mathbb{N}}$, $2_{\mathbb{N}}$}
by (*simp add*: *id-Rel-def incl-Rel-def three nat-omega-simps*)
fix *j* **assume** *j* $\in_\circ$ *set* {*0*, $1_{\mathbb{N}}$, $2_{\mathbb{N}}$}
then consider ‹*j* = *0*› | ‹*j* = $1_{\mathbb{N}}$› | ‹*j* = $2_{\mathbb{N}}$› **by** *auto*
then show *id-Rel* (*F i*)$(\!|j|\!)$ $\in_\circ$ *Q j*
proof *cases*
case *1*
from *i* **show** *?thesis*
unfolding *1*
by
(
subst arr-field-simps(*1*)[*symmetric*],
unfold id-Rel-components Q-def
)
force
next
case *2*
from *i* **show** *?thesis*
unfolding *2*
by
(
subst arr-field-simps(*2*)[*symmetric*],
unfold id-Rel-components Q-def
)

```
        auto
    next
      case 3
      from i show ?thesis
        unfolding 3
        by
          (
            subst arr-field-simps(3)[symmetric],
            unfold id-Rel-components Q-def
          )
          auto
    qed
  qed (auto simp: id-Rel-def cat-Set-cs-intros)
qed
moreover have (∏ₒ iϵₒ set {0, 1_ℕ, 2_ℕ}. Q i) ϵₒ Vset α
proof(rule Limit-vproduct-in-VsetI)
  show set {0, 1_ℕ, 2_ℕ} ϵₒ Vset α unfolding three[symmetric] by simp
  from assms have VPow ((⋃ₒ iϵₒI. F i) ×ₒ (⋃ₒ iϵₒI. F i)) ϵₒ Vset α
    by
      (
        intro
          Limit-VPow-in-VsetI
          Limit-vtimes-in-VsetI
          Limit-vifunion-in-Vset-if-VLambda-in-VsetI
      )
      auto
  then show Q i ϵₒ Vset α if i ϵₒ set {0, 1_ℕ, 2_ℕ} for i
    using that vrange-VLambda
    by (auto intro!: vrange-F-in-Vset simp: Q-def nat-omega-simps)
qed auto
ultimately show ℛₒ (λiϵₒI. cat-Set α(|CId|)(|F i|)) ϵₒ Vset α
  by (meson vsubset-in-VsetI)
qed auto
fix i assume prems: i ϵₒ I
from assms have ℛₒ (VLambda I F) ϵₒ Vset α by (auto simp: vrange-in-VsetI)
moreover from prems have F i ϵₒ ℛₒ (VLambda I F) by auto
ultimately show F i ϵₒ cat-Set α(|Obj|) unfolding cat-Set-components by auto
qed (cs-concl cs-shallow cs-intro: cat-cs-intros assms)+
```

12.2 Product cone and coproduct cocone for the category *Set*

12.2.1 Definition and elementary properties

```
definition ntcf-Set-obj-prod :: V ⇒ V ⇒ (V ⇒ V) ⇒ V
  where ntcf-Set-obj-prod α I F = ntcf-obj-prod-base
    (cat-Set α) I F (∏ₒ iϵₒI. F i) (λi. vprojection-arrow I F i)

definition ntcf-Set-obj-coprod :: V ⇒ V ⇒ (V ⇒ V) ⇒ V
  where ntcf-Set-obj-coprod α I F = ntcf-obj-coprod-base
    (cat-Set α) I F (∐ₒ iϵₒI. F i) (λi. vcinjection-arrow I F i)
```

Components.

```
lemma ntcf-Set-obj-prod-components:
  shows ntcf-Set-obj-prod α I F(|NTMap|) =
    (λiϵₒ:_C I(|Obj|). vprojection-arrow I F i)
    and ntcf-Set-obj-prod α I F(|NTDom|) =
    cf-const (:_C I) (cat-Set α) (∏ₒ iϵₒI. F i)
    and ntcf-Set-obj-prod α I F(|NTCod|) = :→: I F (cat-Set α)
```

and *ntcf-Set-obj-prod* α I $F(\!|NTDGDom|\!)$ = $:_C$ I
and *ntcf-Set-obj-prod* α I $F(\!|NTDGCod|\!)$ = *cat-Set* α
unfolding *ntcf-Set-obj-prod-def ntcf-obj-prod-base-components* **by** *simp-all*

lemma *ntcf-Set-obj-coprod-components*:
shows *ntcf-Set-obj-coprod* α I $F(\!|NTMap|\!)$ =
$(\lambda i \in_\circ :_C I(\!|Obj|\!).$ *vcinjection-arrow* I F $i)$
and *ntcf-Set-obj-coprod* α I $F(\!|NTDom|\!)$ = $:\!\!\rightarrow\!\!: I$ F (*cat-Set* α)
and *ntcf-Set-obj-coprod* α I $F(\!|NTCod|\!)$ =
cf-const ($:_C$ I) (*cat-Set* α) ($\coprod_\circ i \in_\circ I.$ F i)
and *ntcf-Set-obj-coprod* α I $F(\!|NTDGDom|\!)$ = $:_C$ I
and *ntcf-Set-obj-coprod* α I $F(\!|NTDGCod|\!)$ = *cat-Set* α
unfolding *ntcf-Set-obj-coprod-def ntcf-obj-coprod-base-components* **by** *simp-all*

12.2.2 Natural transformation map

mk-VLambda *ntcf-Set-obj-prod-components*(*1*)
|*vsv ntcf-Set-obj-prod-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-Set-obj-prod-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-Set-obj-prod-NTMap-app*[*cat-cs-simps*]|

mk-VLambda *ntcf-Set-obj-coprod-components*(*1*)
|*vsv ntcf-Set-obj-coprod-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-Set-obj-coprod-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-Set-obj-coprod-NTMap-app*[*cat-cs-simps*]|

12.2.3 Product cone for the category *Set* is a universal cone and product cocone for the category *Set* is a universal cocone

lemma (**in** $\mathcal{Z}$) *tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod*:
— See Theorem 5.2 in Chapter Introduction in [6].
assumes *VLambda* I F $\in_\circ$ *Vset* α
shows *ntcf-Set-obj-prod* α I F :
($\prod_\circ i \in_\circ I.$ F i) $<_{CF.\prod}$ $F : I \mapsto\!\!\mapsto_{C\alpha}$ *cat-Set* α
proof(*intro is-cat-obj-prodI is-cat-limitI*)

interpret *Set*: *tm-cf-discrete* α I F ‹*cat-Set* α›
by (*rule tm-cf-discrete-cat-Set-if-VLambda-in-Vset*[*OF assms*])

let *?F* = ‹*ntcf-Set-obj-prod* α I F›

show *cf-discrete* α I F (*cat-Set* α)
by (*auto simp*: *cat-small-discrete-cs-intros*)
show *F-is-cat-cone*: *?F* :
($\prod_\circ i \in_\circ I.$ F i) $<_{CF.cone}$ $:\!\!\rightarrow\!\!: I$ F (*cat-Set* α) : $:_C$ $I \mapsto\!\!\mapsto_{C\alpha}$ *cat-Set* α
unfolding *ntcf-Set-obj-prod-def*
proof(*rule Set.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone*)
show ($\prod_\circ i \in_\circ I.$ F i) $\in_\circ$ *cat-Set* $\alpha(\!|Obj|\!)$
unfolding *cat-Set-components*
by
(
intro
Limit-vproduct-in-Vset-if-VLambda-in-VsetI
Set.tm-cf-discrete-ObjMap-in-Vset
)
auto
qed (*intro vprojection-arrow-is-arr Set.tm-cf-discrete-ObjMap-in-Vset*)

interpret F: *is-cat-cone*
α ‹$\prod_\circ i \in_\circ I.\ F\ i$› ‹$:_C\ I$› ‹*cat-Set* α› ‹$:\!\rightarrow\!:\ I\ F\ (\textit{cat-Set}\ \alpha)$› ‹*?F*›
by (*rule F-is-cat-cone*)

fix $\pi'\ P'$ **assume** *prems*:
$\pi' : P' <_{CF.cone} :\!\rightarrow\!:\ I\ F\ (\textit{cat-Set}\ \alpha) : :_C\ I \mapsto\!\mapsto_{C\alpha}\ \textit{cat-Set}\ \alpha$

let *?π'i* = ‹$\lambda i.\ \pi'(\!|NTMap|\!)(\!|i|\!)$›
let *?up'* = ‹*cat-Set-obj-prod-up I F P' ?π'i*›

interpret π': *is-cat-cone* $\alpha\ P'$ ‹$:_C\ I$› ‹*cat-Set* α› ‹$:\!\rightarrow\!:\ I\ F\ (\textit{cat-Set}\ \alpha)$› π'
by (*rule prems*(*1*))

show $\exists!f'.$
$f' : P' \mapsto_{\textit{cat-Set}\ \alpha} (\prod_\circ i \in_\circ I.\ F\ i) \wedge$
$\pi' = \textit{?F} \cdot_{NTCF} \textit{ntcf-const}\ (:_C\ I)\ (\textit{cat-Set}\ \alpha)\ f'$
proof(*intro ex1I conjI*; (*elim conjE*)*?*)
show *up'*: *?up'* : $P' \mapsto_{\textit{cat-Set}\ \alpha} (\prod_\circ i \in_\circ I.\ F\ i)$
proof(*rule cat-Set-obj-prod-up-cat-Set-is-arr*)
show $P' \in_\circ \textit{cat-Set}\ \alpha(\!|Obj|\!)$ **by** (*auto intro*: *cat-cs-intros cat-lim-cs-intros*)
fix i **assume** $i \in_\circ I$
then show $\pi'(\!|NTMap|\!)(\!|i|\!) : P' \mapsto_{\textit{cat-Set}\ \alpha} F\ i$
by
(
cs-concl **cs-shallow**
cs-simp:
the-cat-discrete-components(*1*)
cat-cs-simps cat-discrete-cs-simps
cs-intro: *cat-cs-intros*
)
qed (*rule assms*)

then have P': $P' \in_\circ \textit{cat-Set}\ \alpha(\!|Obj|\!)$
by (*auto intro*: *cat-cs-intros cat-lim-cs-intros*)

have *π'i-i*: *?π'i i* : $P' \mapsto_{\textit{cat-Set}\ \alpha} F\ i$ **if** $i \in_\circ I$ **for** i
using
π'.ntcf-NTMap-is-arr[*unfolded the-cat-discrete-components*(*1*), *OF that*]
that
by
(
cs-prems **cs-shallow cs-simp**:
cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(*1*)
)

from *cat-Set-obj-prod-up-cat-Set-is-arr*[*OF P' assms*(*1*) *π'i-i*] **have** *π'i*:
cat-Set-obj-prod-up I F P' ?π'i : $P' \mapsto_{\textit{cat-Set}\ \alpha} (\prod_\circ i \in_\circ I.\ F\ i)$.

show $\pi' = \textit{?F} \cdot_{NTCF} \textit{ntcf-const}\ (:_C\ I)\ (\textit{cat-Set}\ \alpha)\ \textit{?up'}$
proof(*rule ntcf-eqI*, *rule π'.is-ntcf-axioms*)

from *F-is-cat-cone π'i* **show**
?F $\cdot_{NTCF}$ *ntcf-const* $(:_C\ I)$ (*cat-Set* α) *?up'* :
cf-const $(:_C\ I)$ (*cat-Set* α) $P' \mapsto_{CF} :\!\rightarrow\!:\ I\ F\ (\textit{cat-Set}\ \alpha) :$
$:_C\ I \mapsto\!\mapsto_{C\alpha}$ *cat-Set* α
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

have *dom-lhs*: $\mathcal{D}_\circ\ (\pi'(\!|NTMap|\!)) = :_C\ I(\!|Obj|\!)$

```
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from F-is-cat-cone π'i have dom-rhs:
    𝒟∘ ((?F ⁃NTCF ntcf-const (:C I) (cat-Set α) ?up')(⌊NTMap⌋)) = :C I(⌊Obj⌋)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  show π'(⌊NTMap⌋) = (?F ⁃NTCF ntcf-const (:C I) (cat-Set α) ?up')(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix i assume prems': i ∈∘ :C I(⌊Obj⌋)
    then have i: i ∈∘ I unfolding the-cat-discrete-components by simp
    have [cat-cs-simps]:
      vprojection-arrow I F i ∘A cat-Set α ?up' = π'(⌊NTMap⌋)(⌊i⌋)
      by
        (
          rule cat-Set-cf-comp-proj-obj-prod-up[
            OF P' assms π'i-i i, symmetric
            ]
        )
        auto
    from π'i prems' show π'(⌊NTMap⌋)(⌊i⌋) =
      (?F ⁃NTCF ntcf-const (:C I) (cat-Set α) ?up')(⌊NTMap⌋)(⌊i⌋)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-Rel-cs-simps cs-intro: cat-cs-intros
        )
  qed (auto simp: cat-cs-intros)

qed simp-all

fix f' assume prems:
  f': P' ↦cat-Set α (∏∘i∈∘I. F i)
  π' = ?F ⁃NTCF ntcf-const (:C I) (cat-Set α) f'
from prems(2) have π'-eq-F-f': π'(⌊NTMap⌋)(⌊i⌋)(⌊ArrVal⌋)(⌊a⌋) =
  (?F ⁃NTCF ntcf-const (:C I) (cat-Set α) f')(⌊NTMap⌋)(⌊i⌋)(⌊ArrVal⌋)(⌊a⌋)
  if i ∈∘ I and a ∈∘ P' for i a
  by simp
have [cat-Set-cs-simps]: π'(⌊NTMap⌋)(⌊i⌋)(⌊ArrVal⌋)(⌊a⌋) = f'(⌊ArrVal⌋)(⌊a⌋)(⌊i⌋)
  if i ∈∘ I and a ∈∘ P' for i a
  using
    π'-eq-F-f'[OF that]
    assms prems that
    vprojection-arrow-is-arr[OF that(1) assms]
  by
    (
      cs-prems cs-shallow
        cs-simp:
          cat-Set-cs-simps
          cat-cs-simps
          vprojection-arrow-ArrVal-app
          the-cat-discrete-components(1)
        cs-intro: cat-Set-cs-intros cat-cs-intros
    )

note f' = cat-Set-is-arrD[OF prems(1)]
note up' = cat-Set-is-arrD[OF up']

interpret f': arr-Set α f' by (rule f'(1))
interpret u': arr-Set α ‹(cat-Set-obj-prod-up I F P' (app (π'(⌊NTMap⌋))))›
```

```
      by (rule up'(1))

    show f' = ?up'
    proof(rule arr-Set-eqI[of α])
      have dom-lhs: 𝒟∘ (f'(|ArrVal|)) = P' by (simp add: cat-Set-cs-simps f')
      have dom-rhs:
        𝒟∘ (cat-Set-obj-prod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)) = P'
        by (simp add: cat-Set-cs-simps up')
      show f'(|ArrVal|) = cat-Set-obj-prod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
        fix a assume prems': a ∈∘ P'
        from prems(1) prems' have f'(|ArrVal|)(|a|) ∈∘ (∏∘ i∈∘ I. F i)
          by (cs-concl cs-shallow cs-intro: cat-Set-cs-intros)
        note f'a = vproductD[OF this]
        from prems' have dom-rhs:
          𝒟∘ (cat-Set-obj-prod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)(|a|)) = I
          by (cs-concl cs-shallow cs-simp: cat-Set-cs-simps)
        show f'(|ArrVal|)(|a|) =
          cat-Set-obj-prod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)(|a|)
        proof(rule vsv-eqI, unfold f'a dom-rhs)
          fix i assume i ∈∘ I
          with prems' show f'(|ArrVal|)(|a|)(|i|) =
            cat-Set-obj-prod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)(|a|)(|i|)
            by (cs-concl cs-shallow cs-simp: cat-Set-cs-simps)
        qed (simp-all add: prems' f'a(1) cat-Set-obj-prod-up-ArrVal-app)
      qed auto
    qed (simp-all add: cat-Set-obj-prod-up-components f' up'(1))

  qed

qed

lemma (in 𝒵) tm-cf-discrete-ntcf-obj-prod-base-is-tm-cat-obj-prod:
  — See Theorem 5.2 in Chapter Introduction in [6].
  assumes VLambda I F ∈∘ Vset α
  shows ntcf-Set-obj-prod α I F :
    (∏∘ i∈∘ I. F i) <CF.tm.∏ F : I ↦↦↦C.tmα cat-Set α
proof(intro is-tm-cat-obj-prodI)
  from assms show tm-cf-discrete α I F (cat-Set α)
    by (rule tm-cf-discrete-cat-Set-if-VLambda-in-Vset)
  show ntcf-Set-obj-prod α I F :
    vproduct I F <CF.lim :→: I F (cat-Set α) ::C I ↦↦↦Cα cat-Set α
    by
      (
        rule is-cat-obj-prodD[
          OF tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod[OF assms]
          ]
      )
qed

lemma (in 𝒵) tm-cf-discrete-ntcf-obj-coprod-base-is-cat-obj-coprod:
  — See Theorem 5.2 in Chapter Introduction in [6].
  assumes VLambda I F ∈∘ Vset α
  shows ntcf-Set-obj-coprod α I F :
    F >CF.∐ (∐∘ i∈∘ I. F i) : I ↦↦↦Cα cat-Set α
proof(intro is-cat-obj-coprodI is-cat-colimitI)

  interpret Set: tm-cf-discrete α I F ‹cat-Set α›
```

```
  by (rule tm-cf-discrete-cat-Set-if-VLambda-in-Vset[OF assms])

let ?F = ‹ntcf-Set-obj-coprod α I F›

show cf-discrete α I F (cat-Set α)
  by (auto simp: cat-small-discrete-cs-intros)
show F-is-cat-cocone: ?F :
  :→: I F (cat-Set α) >_{CF.cocone} (∐_∘ i∈_∘ I. F i) : :_C I ↦↦_{Cα} cat-Set α
  unfolding ntcf-Set-obj-coprod-def
proof(rule Set.tm-cf-discrete-ntcf-obj-coprod-base-is-cat-cocone)
  show (∐_∘ i∈_∘ I. F i) ∈_∘ cat-Set α(|Obj|)
    unfolding cat-Set-components
    by
      (
        intro
          Limit-vdunion-in-Vset-if-VLambda-in-VsetI
          Set.tm-cf-discrete-ObjMap-in-Vset
      )
      auto
qed (intro vcinjection-arrow-is-arr Set.tm-cf-discrete-ObjMap-in-Vset)
then interpret F: is-cat-cocone
  α ‹(∐_∘ i∈_∘ I. F i)› ‹:_C I› ‹cat-Set α› ‹:→: I F (cat-Set α)› ‹?F› .

fix π' P' assume prems:
  π' : :→: I F (cat-Set α) >_{CF.cocone} P' : :_C I ↦↦_{Cα} cat-Set α

let ?π'i = ‹λi. π'(|NTMap|)(|i|)›
let ?up' = ‹cat-Set-obj-coprod-up I F P' ?π'i›

interpret π': is-cat-cocone α P' ‹:_C I› ‹cat-Set α› ‹:→: I F (cat-Set α)› π'
  by (rule prems(1))

show ∃!f'.
  f' : VSigma I F ↦_{cat-Set α} P' ∧
  π' = ntcf-const (:_C I) (cat-Set α) f' ∘_{NTCF} ntcf-Set-obj-coprod α I F
proof(intro ex1I conjI; (elim conjE)?)
  show up': ?up' : (∐_∘ i∈_∘ I. F i) ↦_{cat-Set α} P'
  proof(rule cat-Set-obj-coprod-up-cat-Set-is-arr)
    show P' ∈_∘ cat-Set α(|Obj|)
      by (auto intro: cat-cs-intros cat-lim-cs-intros)
    fix i assume i ∈_∘ I
    then show π'(|NTMap|)(|i|) : F i ↦_{cat-Set α} P'
      by
        (
          cs-concl cs-shallow
            cs-simp:
              cat-cs-simps cat-discrete-cs-simps
              the-cat-discrete-components(1)
            cs-intro: cat-cs-intros
        )
  qed (rule assms)

  then have P': P' ∈_∘ cat-Set α(|Obj|)
    by (auto intro: cat-cs-intros cat-lim-cs-intros)

  have π'i-i: ?π'i i : F i ↦_{cat-Set α} P' if i ∈_∘ I for i
    using
      π'.ntcf-NTMap-is-arr[unfolded the-cat-discrete-components(1), OF that]
```

```
    that
  by
    (
      cs-prems cs-shallow cs-simp:
        cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)
    )
from cat-Set-obj-coprod-up-cat-Set-is-arr[OF P' assms(1) π'i-i] have π'i:
  ?up' : (∐∘ i∈∘ I. F i) ↦_{cat-Set α} P'.

show π' = ntcf-const (:_C I) (cat-Set α) ?up' ∙_{NTCF} ?F
proof(rule ntcf-eqI, rule π'.is-ntcf-axioms)
  from F-is-cat-cocone π'i show
    ntcf-const (:_C I) (cat-Set α) ?up' ∙_{NTCF} ?F :
      :→: I F (cat-Set α) ↦_{CF} cf-const (:_C I) (cat-Set α) P' :
      :_C I ↦↦_{Cα} cat-Set α
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  have dom-lhs: 𝒟∘ (π'(|NTMap|)) = :_C I(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from F-is-cat-cocone π'i have dom-rhs:
    𝒟∘ ((?F ∙_{NTCF} ntcf-const (:_C I) (cat-Set α) ?up')(|NTMap|)) = :_C I(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show π'(|NTMap|) = (ntcf-const (:_C I) (cat-Set α) ?up' ∙_{NTCF} ?F)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix i assume prems': i ∈∘ :_C I(|Obj|)
    then have i: i ∈∘ I unfolding the-cat-discrete-components by simp
    have [cat-cs-simps]:
      ?up' ∘_{A cat-Set α} vcinjection-arrow I F i = π'(|NTMap|)(|i|)
      by
        (
          simp add: cat-Set-cf-comp-coprod-up-vcia[
            OF P' assms π'i-i i, symmetric
            ]
        )
    from π'i prems' show π'(|NTMap|)(|i|) =
      (ntcf-const (:_C I) (cat-Set α) ?up' ∙_{NTCF} ?F)(|NTMap|)(|i|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-Rel-cs-simps cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros)+
qed simp-all

fix f' assume prems:
  f' : (∐∘ i∈∘ I. F i) ↦_{cat-Set α} P'
  π' = ntcf-const (:_C I) (cat-Set α) f' ∙_{NTCF} ?F
from prems(2) have π'-eq-F-f': π'(|NTMap|)(|i|)(|ArrVal|)(|a|) =
  (ntcf-const (:_C I) (cat-Set α) f' ∙_{NTCF} ?F)(|NTMap|)(|i|)(|ArrVal|)(|a|)
  if i ∈∘ I and a ∈∘ P' for i a
  by simp
note f' = cat-Set-is-arrD[OF prems(1)]
note up' = cat-Set-is-arrD[OF up']
interpret f': arr-Set α f' by (rule f'(1))
interpret u': arr-Set α ‹(cat-Set-obj-coprod-up I F P' (app (π'(|NTMap|))))›
  by (rule up'(1))
show f' = ?up'
proof(rule arr-Set-eqI[of α])
  have dom-lhs: 𝒟∘ (f'(|ArrVal|)) = (∐∘ i∈∘ I. F i)
```

```
      by (simp add: cat-Set-cs-simps f')
    have dom-rhs:
      𝒟₀ (cat-Set-obj-coprod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)) =
        (∐₀ i∈₀I. F i)
      by (simp add: cat-Set-cs-simps up')
    show f'(|ArrVal|) = cat-Set-obj-coprod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix ix assume prems': ix ∈₀ (∐₀ i∈₀I. F i)
      then obtain i x where ix-def: ix = ⟨i, x⟩
        and i: i ∈₀ I
        and x: x ∈₀ F i
        by auto
      from assms prems(1) prems' i x show f'(|ArrVal|)(|ix|) =
        cat-Set-obj-coprod-up I F P' (app (π'(|NTMap|)))(|ArrVal|)(|ix|)
        unfolding ix-def prems(2)
        by
          (
            cs-concl cs-shallow
              cs-simp:
                cat-Set-cs-simps cat-cs-simps the-cat-discrete-components(1)
              cs-intro: cat-cs-intros
          )
    qed auto
  qed (simp-all add: cat-Set-obj-coprod-up-components f' up'(1))

 qed

qed

lemma (in 𝒵) ntcf-Set-obj-coprod-is-tm-cat-obj-coprod:
 — See Theorem 5.2 in Chapter Introduction in [6].
 assumes VLambda I F ∈₀ Vset α
 shows ntcf-Set-obj-coprod α I F :
   F >_{CF.tm.∐} (∐₀ i∈₀I. F i) : I ↦↦_{C.tm α} cat-Set α
proof(intro is-tm-cat-obj-coprodI)
 from assms show tm-cf-discrete α I F (cat-Set α)
   by (rule tm-cf-discrete-cat-Set-if-VLambda-in-Vset)
 show ntcf-Set-obj-coprod α I F :
   :→: I F (cat-Set α) >_{CF.colim} VSigma I F : :_C I ↦↦_{Cα} cat-Set α
   by
     (
       rule is-cat-obj-coprodD[
         OF tm-cf-discrete-ntcf-obj-coprod-base-is-cat-obj-coprod[OF assms]
         ]
     )
qed
```

12.3 Equalizer for the category *Set*

12.3.1 Definition and elementary properties

```
abbreviation ntcf-Set-equalizer-map :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
 where ntcf-Set-equalizer-map α a g f i ≡
   (
     i = 𝔞_{PL2} ?
       incl-Set (vequalizer a g f) a :
       g ∘_{A cat-Set α} incl-Set (vequalizer a g f) a
   )
```

definition *ntcf-Set-equalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-Set-equalizer* α a b g f = *ntcf-equalizer-base*
    (*cat-Set* α) a b g f (*vequalizer* a g f) (*ntcf-Set-equalizer-map* α a g f)

Components.

context
  fixes a g f α :: V
begin

lemmas *ntcf-Set-equalizer-components* =
  ntcf-equalizer-base-components[
      where $\mathfrak{C}$=‹*cat-Set* α›
        and *e*=‹*ntcf-Set-equalizer-map* α a g f›
        and *E*=‹*vequalizer* a g f›
        and $\mathfrak{a}$=a **and** $\mathfrak{g}$=g **and** $\mathfrak{f}$=f,
        folded ntcf-Set-equalizer-def
    ]

end

12.3.2 Natural transformation map

mk-VLambda *ntcf-Set-equalizer-components*(*1*)
  |*vsv ntcf-Set-equalizer-NTMap-vsv*[*cat-Set-cs-intros*]|
  |*vdomain ntcf-Set-equalizer-NTMap-vdomain*[*cat-Set-cs-simps*]|
  |*app ntcf-Set-equalizer-NTMap-app*|

lemma *ntcf-Set-equalizer-2-NTMap-app-*$\mathfrak{a}$[*cat-Set-cs-simps*]:
  assumes x = $\mathfrak{a}_{PL2}$
  shows
    ntcf-Set-equalizer α a b g f(⁅*NTMap*⁆)(⁅x⁆) =
      incl-Set (*vequalizer* a g f) a
  unfolding *assms the-cat-parallel-2-components*(*1*) *ntcf-Set-equalizer-components*
  by *simp*

lemma *ntcf-Set-equalizer-2-NTMap-app-*$\mathfrak{b}$[*cat-Set-cs-simps*]:
  assumes x = $\mathfrak{b}_{PL2}$
  shows
    ntcf-Set-equalizer α a b g f(⁅*NTMap*⁆)(⁅x⁆) =
      g $\circ_{A\,cat\text{-}Set\ \alpha}$ *incl-Set* (*vequalizer* a g f) a
  unfolding *assms the-cat-parallel-2-components*(*1*) *ntcf-Set-equalizer-components*
  using *cat-PL2-ineq*
  by *auto*

12.3.3 Equalizer for the category *Set* is an equalizer

lemma (**in** $\mathcal{Z}$) *ntcf-Set-equalizer-2-is-cat-equalizer-2*:
  assumes $\mathfrak{g}$: $\mathfrak{a}$ $\mapsto_{cat\text{-}Set\ \alpha}$ $\mathfrak{b}$ **and** $\mathfrak{f}$: $\mathfrak{a}$ $\mapsto_{cat\text{-}Set\ \alpha}$ $\mathfrak{b}$
  shows *ntcf-Set-equalizer* α $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$:
    vequalizer $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{f}$ $<_{CF.eq}$ ($\mathfrak{a}$,$\mathfrak{b}$,$\mathfrak{g}$,$\mathfrak{f}$) : $\uparrow\uparrow_C$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof(*intro is-cat-equalizer-2I is-cat-equalizerI is-cat-limitI*)

  let *?II-II* = ‹$\uparrow\uparrow\to\uparrow\uparrow_{CF}$ (*cat-Set* α) $\mathfrak{a}_{PL2}$ $\mathfrak{b}_{PL2}$ $\mathfrak{g}_{PL}$ $\mathfrak{f}_{PL}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$›
    and *?II* = ‹$\uparrow\uparrow_C$ $\mathfrak{a}_{PL2}$ $\mathfrak{b}_{PL2}$ $\mathfrak{g}_{PL}$ $\mathfrak{f}_{PL}$›

  note $\mathfrak{g}$ = *cat-Set-is-arrD*[*OF assms*(*1*)]
  interpret $\mathfrak{g}$: *arr-Set* α $\mathfrak{g}$

```
  rewrites 𝔤(|ArrDom|) = 𝔞 and 𝔤(|ArrCod|) = 𝔟
  by (rule 𝔤(1)) (simp-all add: 𝔤)
note 𝔣 = cat-Set-is-arrD[OF assms(2)]
interpret 𝔣: arr-Set α 𝔣
  rewrites 𝔣(|ArrDom|) = 𝔞 and 𝔣(|ArrCod|) = 𝔟
  by (rule 𝔣(1)) (simp-all add: 𝔣)

note [cat-Set-cs-intros] = 𝔤.arr-Set-ArrDom-in-Vset 𝔣.arr-Set-ArrCod-in-Vset

let ?incl = ‹incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞›

show 𝔞𝔟𝔤𝔣-is-cat-cone: ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 :
  vequalizer 𝔞 𝔤 𝔣 <CF.cone ?II-II : ?II ↦↦CFα cat-Set α
  unfolding ntcf-Set-equalizer-def
proof
  (
    intro
      category.cat-ntcf-equalizer-base-is-cat-cone
      category.cat-cf-parallel-2-cat-equalizer
  )
  from assms show
    (𝔟PL2 = 𝔞PL2 ? ?incl : 𝔤 ∘A cat-Set α ?incl) :
      vequalizer 𝔞 𝔤 𝔣 ↦cat-Set α 𝔟
    by
      (
        cs-concl
          cs-simp: V-cs-simps
          cs-intro:
            V-cs-intros cat-Set-cs-intros cat-cs-intros
            cat-PL2-ineq[symmetric]
      )
  show
    (𝔟PL2 = 𝔞PL2 ? ?incl : 𝔤 ∘A cat-Set α ?incl) =
      𝔤 ∘A cat-Set α (𝔞PL2 = 𝔞PL2 ? ?incl : 𝔤 ∘A cat-Set α ?incl)
    by
      (
        cs-concl
          cs-simp: V-cs-simps
          cs-intro:
            V-cs-intros cat-Set-cs-intros cat-cs-intros
            cat-PL2-ineq[symmetric]
      )
  from assms show
    (𝔟PL2 = 𝔞PL2 ? ?incl : 𝔤 ∘A cat-Set α ?incl) =
      𝔣 ∘A cat-Set α (𝔞PL2 = 𝔞PL2 ? ?incl : 𝔤 ∘A cat-Set α ?incl)
    by
      (
        cs-concl
          cs-simp: V-cs-simps cat-Set-incl-Set-commute
          cs-intro: V-cs-intros cat-PL2-ineq[symmetric]
      )
qed
  (
    cs-concl
      cs-intro: cat-cs-intros V-cs-intros cat-Set-cs-intros assms
      cs-simp: V-cs-simps cat-cs-simps cat-Set-components(1)
  )+
```

```
interpret 𝔞𝔟𝔤𝔣: is-cat-cone
  α ‹vequalizer 𝔞 𝔤 𝔣› ?II ‹cat-Set α› ?II-II ‹ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣›
  by (rule 𝔞𝔟𝔤𝔣-is-cat-cone)

show ∃!f'.
  f' : r' ↦_cat-Set α vequalizer 𝔞 𝔤 𝔣 ∧
  u' = ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∙_NTCF ntcf-const ?II (cat-Set α) f'
  if u' : r' <_CF.cone ?II-II : ?II ↦↦_Cα cat-Set α for u' r'
proof–

  interpret u': is-cat-cone α r' ?II ‹cat-Set α› ?II-II u' by (rule that(1))

  have 𝔞_PL2 ∈_∘ ?II(|Obj|)
    unfolding the-cat-parallel-2-components(1) by simp
  from
    u'.ntcf-NTMap-is-arr[OF this]
    𝔞𝔟𝔤𝔣.NTDom.HomCod.cat-cf-parallel-2-cat-equalizer[OF assms]
  have u'-𝔞_PL-is-arr: u'(|NTMap|)(|𝔞_PL2|) : r' ↦_cat-Set α 𝔞
    by (cs-prems-atom-step cat-cs-simps)
      (
        cs-prems
          cs-simp: cat-parallel-cs-simps
          cs-intro:
            cat-parallel-cs-intros
            cat-cs-intros
            category.cat-cf-parallel-2-cat-equalizer
      )
  note u'-𝔞_PL = cat-Set-is-arrD[OF u'-𝔞_PL-is-arr]
  interpret u'-𝔞_PL: arr-Set α ‹u'(|NTMap|)(|𝔞_PL2|)› by (rule u'-𝔞_PL(1))

  have 𝔟_PL2 ∈_∘ ?II(|Obj|)
    by (cs-concl cs-shallow cs-intro: cat-parallel-cs-intros)

  from
    u'.ntcf-NTMap-is-arr[OF this]
    𝔞𝔟𝔤𝔣.NTDom.HomCod.cat-cf-parallel-2-cat-equalizer[OF assms]
  have u'(|NTMap|)(|𝔟_PL2|) : r' ↦_cat-Set α 𝔟
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cat-parallel-cs-simps
          cs-intro: cat-parallel-cs-intros
      )

  note u'-𝔤u' = cat-cone-cf-par-2-eps-NTMap-app(1)[OF that(1) assms]

  define q where q = [u'(|NTMap|)(|𝔞_PL2|)(|ArrVal|), r', vequalizer 𝔞 𝔤 𝔣]_∘

  have q-components[cat-Set-cs-simps]:
    q(|ArrVal|) = u'(|NTMap|)(|𝔞_PL2|)(|ArrVal|)
    q(|ArrDom|) = r'
    q(|ArrCod|) = vequalizer 𝔞 𝔤 𝔣
    unfolding q-def arr-field-simps by (simp-all add: nat-omega-simps)

  from cat-cone-cf-par-2-eps-NTMap-app[OF that(1) assms] have 𝔤u'-eq-𝔣u':
    (𝔤 ∘_A cat-Set α u'(|NTMap|)(|𝔞_PL2|))(|ArrVal|)(|x|) =
      (𝔣 ∘_A cat-Set α u'(|NTMap|)(|𝔞_PL2|))(|ArrVal|)(|x|)
    for x
```

```
  by simp

show ?thesis
proof(intro ex1I conjI; (elim conjE)?)

  have u'-NTMap-vrange: ℛ∘ (u'(|NTMap|)(|𝔞PL2|)(|ArrVal|)) ⊆∘ vequalizer 𝔞 𝔤 𝔣
  proof(rule vsubsetI)
    fix y assume prems: y ∈∘ ℛ∘ (u'(|NTMap|)(|𝔞PL2|)(|ArrVal|))
    then obtain x where x: x ∈∘ 𝒟∘ (u'(|NTMap|)(|𝔞PL2|)(|ArrVal|))
      and y-def: y = u'(|NTMap|)(|𝔞PL2|)(|ArrVal|)(|x|)
      by (blast dest: u'-𝔞PL.ArrVal.vrange-atD)
    have x: x ∈∘ r'
      by (use x u'-𝔞PL-is-arr in ‹cs-prems cs-shallow cs-simp: cat-cs-simps›)
    from 𝔤u'-eq-𝔣u'[of x] assms x u'-𝔞PL-is-arr have [simp]:
      𝔤(|ArrVal|)(|u'(|NTMap|)(|𝔞PL2|)(|ArrVal|)(|x|)|) =
        𝔣(|ArrVal|)(|u'(|NTMap|)(|𝔞PL2|)(|ArrVal|)(|x|)|)
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from prems u'-𝔞PL.arr-Set-ArrVal-vrange[unfolded u'-𝔞PL] show
      y ∈∘ vequalizer 𝔞 𝔤 𝔣
      by (intro vequalizerI, unfold y-def) auto
  qed

  show q-is-arr: q : r' ↦cat-Set α vequalizer 𝔞 𝔤 𝔣
  proof(intro cat-Set-is-arrI arr-SetI)
    show q(|ArrCod|) ∈∘ Vset α
      by (auto simp: q-components intro: cat-cs-intros cat-lim-cs-intros)
  qed
    (
      auto
        simp:
          cat-Set-cs-simps nat-omega-simps
          u'-𝔞PL
          q-def
          u'-NTMap-vrange
          𝔞𝔟𝔤𝔣.NTDom.HomCod.cat-in-Obj-in-Vset
        intro: cat-cs-intros cat-lim-cs-intros
    )

  from q-is-arr have 𝔞-q:
    incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘A cat-Set α q : r' ↦cat-Set α 𝔞
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-Set-components(1)
          cs-intro: V-cs-intros cat-cs-intros cat-Set-cs-intros
      )
  interpret arr-Set α ‹incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘A cat-Set α q›
    using 𝔞-q by (auto dest: cat-Set-is-arrD)

  show u' = ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∙NTCF ntcf-const ?II (cat-Set α) q
  proof(rule ntcf-eqI)
    from q-is-arr show
      ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∙NTCF ntcf-const ?II (cat-Set α) q :
        cf-const ?II (cat-Set α) r' ↦CF
        ?II-II : ?II ↦↦Cα cat-Set α
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    have dom-lhs: 𝒟∘ (u'(|NTMap|)) = ?II(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
```

```
from q-is-arr have dom-rhs:
  𝒟∘
    (
      (ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ⊙NTCF
      ntcf-const ?II (cat-Set α) q
    )(⦇NTMap⦈) =  ?II(⦇Obj⦈)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
show u′(⦇NTMap⦈) =
  (
    ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ⊙NTCF ntcf-const ?II (cat-Set α) q
  )(⦇NTMap⦈)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  show vsv ((
    ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ⊙NTCF ntcf-const ?II (cat-Set α) q
    )(⦇NTMap⦈))
    by (cs-concl cs-intro: cat-cs-intros)
  fix a assume prems: a ∈∘ ?II(⦇Obj⦈)
  have [symmetric, cat-Set-cs-simps]:
    u′(⦇NTMap⦈)(⦇𝔞PL2⦈) = incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘A cat-Set α q
  proof(rule arr-Set-eqI[of α])
    from u′-𝔞PL-is-arr have dom-lhs: 𝒟∘ (u′(⦇NTMap⦈)(⦇𝔞PL2⦈)(⦇ArrVal⦈)) = r′
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros
        )
    from 𝔞-q have dom-rhs:
      𝒟∘ ((incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘A cat-Set α q)(⦇ArrVal⦈)) = r′
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros
        )
    show u′(⦇NTMap⦈)(⦇𝔞PL2⦈)(⦇ArrVal⦈) =
      (incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘A cat-Set α q)(⦇ArrVal⦈)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume prems: a ∈∘ r′
      with u′-NTMap-vrange dom-lhs u′-𝔞PL.ArrVal.vsv-vimageI2 have
        u′(⦇NTMap⦈)(⦇𝔞PL2⦈)(⦇ArrVal⦈)(⦇a⦈) ∈∘ vequalizer 𝔞 𝔤 𝔣
        by blast
      with prems q-is-arr u′-𝔞PL-is-arr show
        u′(⦇NTMap⦈)(⦇𝔞PL2⦈)(⦇ArrVal⦈)(⦇a⦈) =
          (incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘A cat-Set α q)(⦇ArrVal⦈)(⦇a⦈)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-Set-cs-simps cat-cs-simps
              cs-intro: V-cs-intros cat-cs-intros cat-Set-cs-intros
          )
    qed auto
  qed
    (
      use u′-𝔞PL 𝔞-q in ‹
        cs-concl cs-shallow
          cs-intro: cat-Set-is-arrD(1) cs-simp: cat-cs-simps
        ›
    )+
  from q-is-arr have u′-NTMap-app-I: u′(⦇NTMap⦈)(⦇𝔞PL2⦈) =
```

```
    (
      ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∘_NTCF ntcf-const ?II (cat-Set α) q
    )(⟦NTMap⟧)(⟦𝔞_PL2⟧)
    by
      (
        cs-concl
          cs-intro: cat-cs-intros cat-parallel-cs-intros
          cs-simp: cat-Set-cs-simps cat-cs-simps V-cs-simps
      )
  from q-is-arr assms have u'-NTMap-app-sI: u'(⟦NTMap⟧)(⟦𝔟_PL2⟧) =
    (
      ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∘_NTCF ntcf-const ?II (cat-Set α) q
    )(⟦NTMap⟧)(⟦𝔟_PL2⟧)
    by
      (
        cs-concl
          cs-simp: cat-Set-cs-simps cat-cs-simps u'-𝔤u'
          cs-intro:
            V-cs-intros
            cat-cs-intros
            cat-Set-cs-intros
            cat-parallel-cs-intros
      )
  from prems consider ‹a = 𝔞_PL2› | ‹a = 𝔟_PL2›
    by (elim the-cat-parallel-2-ObjE)
  then show
    u'(⟦NTMap⟧)(⟦a⟧) =
      (
        ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∘_NTCF
        ntcf-const ?II (cat-Set α) q
      )(⟦NTMap⟧)(⟦a⟧)
    by cases (simp-all add: u'-NTMap-app-I u'-NTMap-app-sI)
qed auto
qed (simp-all add: u'.is-ntcf-axioms)

fix f' assume prems:
  f' : r' ↦_cat-Set α vequalizer 𝔞 𝔤 𝔣
  u' = ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∘_NTCF ntcf-const ?II (cat-Set α) f'
from prems(2) have u'-NTMap-app:
  u'(⟦NTMap⟧)(⟦x⟧) =
    (ntcf-Set-equalizer α 𝔞 𝔟 𝔤 𝔣 ∘_NTCF
    ntcf-const ?II (cat-Set α) f')(⟦NTMap⟧)(⟦x⟧)
  for x
  by simp
have u'-f':
  u'(⟦NTMap⟧)(⟦𝔞_PL2⟧) = incl-Set (vequalizer 𝔞 𝔤 𝔣) 𝔞 ∘_A cat-Set α f'
  using u'-NTMap-app[of 𝔞_PL2] prems(1)
  by
    (
      cs-prems
        cs-simp: cat-cs-simps
        cs-intro: cat-cs-intros cat-parallel-cs-intros
    )
    (
      cs-prems cs-shallow
        cs-simp: cat-Set-cs-simps cs-intro: cat-parallel-cs-intros
    )
```

```
    note f' = cat-Set-is-arrD[OF prems(1)]
    note q = cat-Set-is-arrD[OF q-is-arr]

    interpret f': arr-Set α f' using prems(1) by (auto dest: cat-Set-is-arrD)
    interpret q: arr-Set α q using q by (auto dest: cat-Set-is-arrD)

    show f' = q
    proof(rule arr-Set-eqI[of α])
      have dom-lhs: 𝒟₀ (f'(|ArrVal|)) = r' by (simp add: cat-Set-cs-simps f')
      from q-is-arr have dom-rhs: 𝒟₀ (q(|ArrVal|)) = r'
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cs-intro: cat-Set-cs-intros
          )
      show f'(|ArrVal|) = q(|ArrVal|)
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
        fix i assume i ∈₀ r'
        with prems(1) show f'(|ArrVal|)(|i|) = q(|ArrVal|)(|i|)
          by
            (
              cs-concl
                cs-simp:
                  cat-Set-cs-simps cat-cs-simps
                  q-components u'-f' cat-Set-components(1)
                cs-intro: V-cs-intros cat-cs-intros cat-Set-cs-intros
            )
      qed auto
    qed
      (
        use prems(1) q-is-arr in ‹
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: q cat-Set-is-arrD
          ›
      )+
  qed
qed

qed (auto intro: assms)
```

12.4 The category *Set* is small-complete

```
lemma (in 𝒵) cat-small-complete-cat-Set: cat-small-complete α (cat-Set α)
  — This lemma appears as a remark on page 113 in [9].
proof(rule category.cat-small-complete-if-eq-and-obj-prod)
  show ∃E ε. ε : E <_{CF.eq} (𝔞,𝔟,𝔤,𝔣) : ↑↑_C ↦↦_{Cα} cat-Set α
    if 𝔣 : 𝔞 ↦_{cat-Set α} 𝔟 and 𝔤 : 𝔞 ↦_{cat-Set α} 𝔟 for 𝔞 𝔟 𝔤 𝔣
    using ntcf-Set-equalizer-2-is-cat-equalizer-2[OF that(2,1)] by auto
  show ∃P π. π : P <_{CF.∏} A : I ↦↦_{Cα} cat-Set α
    if tm-cf-discrete α I A (cat-Set α) for A I
  proof(intro exI, rule tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod)
    interpret tm-cf-discrete α I A ‹cat-Set α› by (rule that)
    show VLambda I A ∈₀ Vset α by (rule tm-cf-discrete-ObjMap-in-Vset)
  qed
qed (rule category-cat-Set)
```

13 Adjoints

13.1 Background

named-theorems *adj-cs-simps*
named-theorems *adj-cs-intros*
named-theorems *adj-field-simps*

definition *AdjLeft* :: *V* **where** [*adj-field-simps*]: *AdjLeft* = *0*
definition *AdjRight* :: *V* **where** [*adj-field-simps*]: *AdjRight* = $1_{\mathbb{N}}$
definition *AdjNT* :: *V* **where** [*adj-field-simps*]: *AdjNT* = $2_{\mathbb{N}}$

13.2 Definition and elementary properties

See subsection 2.1 in [4] or Chapter IV-1 in [9].

locale *is-cf-adjunction* =
  $\mathcal{Z}$ α +
  vfsequence Φ +
  L: *category* α $\mathfrak{C}$ +
  R: *category* α $\mathfrak{D}$ +
  LR: *is-functor* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ +
  RL: *is-functor* α $\mathfrak{D}$ $\mathfrak{C}$ $\mathfrak{G}$ +
  NT: *is-iso-ntcf*
    α
    ‹*op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{D}$›
    ‹*cat-Set* α›
    ‹$Hom_{O.C\alpha}\mathfrak{D}(\mathfrak{F}-,-)$›
    ‹$Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$›
    ‹$\Phi(\!|AdjNT|\!)$›
    for α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ Φ +
  assumes *cf-adj-length*[*adj-cs-simps*]: *vcard* Φ = $3_{\mathbb{N}}$
    and *cf-adj-AdjLeft*[*adj-cs-simps*]: $\Phi(\!|AdjLeft|\!)$ = $\mathfrak{F}$
    and *cf-adj-AdjRight*[*adj-cs-simps*]: $\Phi(\!|AdjRight|\!)$ = $\mathfrak{G}$

syntax *-is-cf-adjunction* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
  (‹(- : - ⇌$_{CF}$ - : - ⇌⇌$_{C1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cf-adjunction* ⇌ *is-cf-adjunction*
translations Φ : $\mathfrak{F}$ ⇌$_{CF}$ $\mathfrak{G}$: $\mathfrak{C}$ ⇌⇌$_{C\alpha}$ $\mathfrak{D}$ ⇌
  CONST is-cf-adjunction α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ Φ

lemmas [*adj-cs-simps*] =
  is-cf-adjunction.cf-adj-length
  is-cf-adjunction.cf-adj-AdjLeft
  is-cf-adjunction.cf-adj-AdjRight

Components.

lemma *cf-adjunction-components*[*adj-cs-simps*]:
  $[\mathfrak{F}, \mathfrak{G}, \varphi]_\circ(\!|AdjLeft|\!)$ = $\mathfrak{F}$
  $[\mathfrak{F}, \mathfrak{G}, \varphi]_\circ(\!|AdjRight|\!)$ = $\mathfrak{G}$
  $[\mathfrak{F}, \mathfrak{G}, \varphi]_\circ(\!|AdjNT|\!)$ = φ
  unfolding *AdjLeft-def AdjRight-def AdjNT-def*
  by (*simp-all add*: *nat-omega-simps*)

Rules.

lemma (**in** *is-cf-adjunction*) *is-cf-adjunction-axioms'*[*adj-cs-intros*]:
  assumes α' = α **and** $\mathfrak{C}'$ = $\mathfrak{C}$ **and** $\mathfrak{D}'$ = $\mathfrak{D}$ **and** $\mathfrak{F}'$ = $\mathfrak{F}$ **and** $\mathfrak{G}'$ = $\mathfrak{G}$
  shows Φ : $\mathfrak{F}'$ ⇌$_{CF}$ $\mathfrak{G}'$: $\mathfrak{C}'$ ⇌⇌$_{C\alpha'}$ $\mathfrak{D}'$

unfolding *assms* **by** (*rule is-cf-adjunction-axioms*)

lemmas (**in** *is-cf-adjunction*) [*adj-cs-intros*] = *is-cf-adjunction-axioms*

mk-ide rf *is-cf-adjunction-def*[*unfolded is-cf-adjunction-axioms-def*]
|*intro is-cf-adjunctionI*|
|*dest is-cf-adjunctionD*[*dest*]|
|*elim is-cf-adjunctionE*[*elim*]|

lemmas [*adj-cs-intros*] = *is-cf-adjunctionD*(*3–6*)

lemma (**in** *is-cf-adjunction*) *cf-adj-is-iso-ntcf'*:
assumes $\mathfrak{F}' = \mathit{Hom}_{O.C\alpha}\mathfrak{D}(\mathfrak{F}-,-)$
and $\mathfrak{G}' = \mathit{Hom}_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$
and $\mathfrak{A}' = \mathit{op\text{-}cat}\ \mathfrak{C} \times_C \mathfrak{D}$
and $\mathfrak{B}' = \mathit{cat\text{-}Set}\ \alpha$
shows $\Phi(\!|\mathit{AdjNT}|\!) : \mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*auto intro*: *cat-cs-intros*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adj-is-iso-ntcf'*

lemma *cf-adj-eqI*:
assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
and $\Phi' : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G}' : \mathfrak{C}' \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}'$
and $\mathfrak{C} = \mathfrak{C}'$
and $\mathfrak{D} = \mathfrak{D}'$
and $\mathfrak{F} = \mathfrak{F}'$
and $\mathfrak{G} = \mathfrak{G}'$
and $\Phi(\!|\mathit{AdjNT}|\!) = \Phi'(\!|\mathit{AdjNT}|\!)$
shows $\Phi = \Phi'$
proof–
interpret Φ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}\ \Phi$ **by** (*rule assms*(*1*))
interpret Φ': *is-cf-adjunction* $\alpha\ \mathfrak{C}'\ \mathfrak{D}'\ \mathfrak{F}'\ \mathfrak{G}'\ \Phi'$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom*: $\mathcal{D}_\circ\ \Phi = 3_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *V-cs-simps adj-cs-simps*)
show $\mathcal{D}_\circ\ \Phi = \mathcal{D}_\circ\ \Phi'$
by (*cs-concl* **cs-shallow cs-simp**: *V-cs-simps adj-cs-simps dom*)
from *assms*(*4–7*) **have** *sup*:
$\Phi(\!|\mathit{AdjLeft}|\!) = \Phi'(\!|\mathit{AdjLeft}|\!)$
$\Phi(\!|\mathit{AdjRight}|\!) = \Phi'(\!|\mathit{AdjRight}|\!)$
$\Phi(\!|\mathit{AdjNT}|\!) = \Phi'(\!|\mathit{AdjNT}|\!)$
by (*simp-all add*: *adj-cs-simps*)
show $a \in_\circ \mathcal{D}_\circ\ \Phi \Longrightarrow \Phi(\!|a|\!) = \Phi'(\!|a|\!)$ **for** *a*
by (*unfold dom*, *elim-in-numeral*, *insert sup*)
(*auto simp*: *adj-field-simps*)
qed (*auto simp*: Φ.*L.vsv-axioms* Φ'.*vsv-axioms*)
qed

13.3 Opposite adjunction

13.3.1 Definition and elementary properties

See [7] for further information.

abbreviation *op-cf-adj-nt* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *op-cf-adj-nt* $\mathfrak{C}\ \mathfrak{D}\ \varphi \equiv \mathit{inv\text{-}ntcf}\ (\mathit{bnt\text{-}flip}\ (\mathit{op\text{-}cat}\ \mathfrak{C})\ \mathfrak{D}\ \varphi)$

definition *op-cf-adj* :: $V \Rightarrow V$
  where *op-cf-adj* Φ =
    [
      op-cf ($\Phi(\!|AdjRight|\!)$),
      op-cf ($\Phi(\!|AdjLeft|\!)$),
      op-cf-adj-nt ($\Phi(\!|AdjLeft|\!)(\!|HomDom|\!)$) ($\Phi(\!|AdjLeft|\!)(\!|HomCod|\!)$) ($\Phi(\!|AdjNT|\!)$)
    ]$_{\circ}$

lemma *op-cf-adj-components*:
  shows *op-cf-adj* $\Phi(\!|AdjLeft|\!)$ = *op-cf* ($\Phi(\!|AdjRight|\!)$)
    and *op-cf-adj* $\Phi(\!|AdjRight|\!)$ = *op-cf* ($\Phi(\!|AdjLeft|\!)$)
    and *op-cf-adj* $\Phi(\!|AdjNT|\!)$ =
      op-cf-adj-nt ($\Phi(\!|AdjLeft|\!)(\!|HomDom|\!)$) ($\Phi(\!|AdjLeft|\!)(\!|HomCod|\!)$) ($\Phi(\!|AdjNT|\!)$)
  unfolding *op-cf-adj-def adj-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma (**in** *is-cf-adjunction*) *op-cf-adj-components*:
  shows *op-cf-adj* $\Phi(\!|AdjLeft|\!)$ = *op-cf* $\mathfrak{G}$
    and *op-cf-adj* $\Phi(\!|AdjRight|\!)$ = *op-cf* $\mathfrak{F}$
    and *op-cf-adj* $\Phi(\!|AdjNT|\!)$ = *inv-ntcf* (*bnt-flip* (*op-cat* $\mathfrak{C}$) $\mathfrak{D}$ ($\Phi(\!|AdjNT|\!)$))
  unfolding *op-cf-adj-components* **by** (*simp-all add*: *cat-cs-simps adj-cs-simps*)

lemmas [*cat-op-simps*] = *is-cf-adjunction.op-cf-adj-components*

The opposite adjunction is an adjunction.

lemma (**in** *is-cf-adjunction*) *is-cf-adjunction-op*:
  — See comments in subsection 2.1 in [4].
  op-cf-adj Φ : *op-cf* $\mathfrak{G}$ $\rightleftharpoons_{CF}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{D}$ $\rightleftharpoons\rightleftharpoons_{C\alpha}$ *op-cat* $\mathfrak{C}$
proof(*intro is-cf-adjunctionI*, *unfold cat-op-simps*, *unfold op-cf-adj-components*)
  show *vfsequence* (*op-cf-adj* Φ) **unfolding** *op-cf-adj-def* **by** *simp*
  show *vcard* (*op-cf-adj* Φ) = $3_{\mathbb{N}}$
    unfolding *op-cf-adj-def* **by** (*simp add*: *nat-omega-simps*)
  note *adj* = *is-cf-adjunctionD*[*OF is-cf-adjunction-axioms*]
  from *adj* **have** *f-φ*: *bnt-flip* (*op-cat* $\mathfrak{C}$) $\mathfrak{D}$ ($\Phi(\!|AdjNT|\!)$) :
    $Hom_{O.C\alpha}$*op-cat* $\mathfrak{D}(-,$*op-cf* $\mathfrak{F}-) \mapsto_{CF.iso} Hom_{O.C\alpha}$*op-cat* $\mathfrak{C}($*op-cf* $\mathfrak{G}-,-)$:
    $\mathfrak{D} \times_C$ *op-cat* $\mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)
  show *op-cf-adj-nt* $\mathfrak{C}$ $\mathfrak{D}$ ($\Phi(\!|AdjNT|\!)$) :
    $Hom_{O.C\alpha}$*op-cat* $\mathfrak{C}($*op-cf* $\mathfrak{G}-,-) \mapsto_{CF.iso} Hom_{O.C\alpha}$*op-cat* $\mathfrak{D}(-,$*op-cf* $\mathfrak{F}-)$:
    $\mathfrak{D} \times_C$ *op-cat* $\mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
    by (*rule CZH-ECAT-NTCF.iso-ntcf-is-iso-arr*(*1*)[*OF f-φ*])
qed (*auto intro*: *cat-cs-intros cat-op-intros*)

lemmas *is-cf-adjunction-op* =
  is-cf-adjunction.is-cf-adjunction-op

lemma (**in** *is-cf-adjunction*) *is-cf-adjunction-op′*[*cat-op-intros*]:
  assumes $\mathfrak{G}'$ = *op-cf* $\mathfrak{G}$
    and $\mathfrak{F}'$ = *op-cf* $\mathfrak{F}$
    and $\mathfrak{D}'$ = *op-cat* $\mathfrak{D}$
    and $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
  shows *op-cf-adj* Φ : $\mathfrak{G}' \rightleftharpoons_{CF} \mathfrak{F}' : \mathfrak{D}' \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cf-adjunction-op*)

lemmas [*cat-op-intros*] = *is-cf-adjunction.is-cf-adjunction-op′*

The operation of taking the opposite adjunction is an involution.

lemma (**in** *is-cf-adjunction*) *cf-adjunction-op-cf-adj-op-cf-adj*[*cat-op-simps*]:
  op-cf-adj (*op-cf-adj* Φ) = Φ
proof(*rule cf-adj-eqI*)
  show Φ′: *op-cf-adj* (*op-cf-adj* Φ) : 𝔉 ⇌$_{CF}$ 𝔊 : ℭ ⇌⇌$_{C\alpha}$ 𝔇
  proof(*intro is-cf-adjunctionI*)
    show *vfsequence* (*op-cf-adj* (*op-cf-adj* Φ)) **unfolding** *op-cf-adj-def* **by** *simp*
    from *is-cf-adjunction-axioms* **show** *op-cf-adj* (*op-cf-adj* Φ)(|*AdjNT*|) :
      Hom$_{O.C\alpha}$𝔇(𝔉−,−) ↦$_{CF.iso}$ *Hom*$_{O.C\alpha}$ℭ(−,𝔊−) :
      op-cat ℭ ×$_C$ 𝔇 ↦↦$_{C\alpha}$ *cat-Set* α
      by
        (
          cs-concl **cs-shallow**
            cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros*
            cs-simp: *cat-cs-simps cat-op-simps*
)
    show *vcard* (*op-cf-adj* (*op-cf-adj* Φ)) = $\mathcal{3}_{\mathbb{N}}$
      unfolding *op-cf-adj-def* **by** (*simp add*: *nat-omega-simps*)
    from *is-cf-adjunction-axioms* **show** *op-cf-adj* (*op-cf-adj* Φ)(|*AdjLeft*|) = 𝔉
      by (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)
    from *is-cf-adjunction-axioms* **show** *op-cf-adj* (*op-cf-adj* Φ)(|*AdjRight*|) = 𝔊
      by (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)
  qed (*auto intro*: *cat-cs-intros*)
  interpret Φ′: *is-cf-adjunction* α ℭ 𝔇 𝔉 𝔊 ‹*op-cf-adj* (*op-cf-adj* Φ)›
    by (*rule* Φ′)
  show *op-cf-adj* (*op-cf-adj* Φ)(|*AdjNT*|) = Φ(|*AdjNT*|)
  proof(*rule ntcf-eqI*)
    show *op-op*-Φ:
      op-cf-adj (*op-cf-adj* Φ)(|*AdjNT*|) :
        Hom$_{O.C\alpha}$𝔇(𝔉−,−) ↦$_{CF}$ *Hom*$_{O.C\alpha}$ℭ(−,𝔊−) :
        op-cat ℭ ×$_C$ 𝔇 ↦↦$_{C\alpha}$ *cat-Set* α
      by (*rule* Φ′.*NT.is-ntcf-axioms*)
    show Φ: Φ(|*AdjNT*|) :
      Hom$_{O.C\alpha}$𝔇(𝔉−,−) ↦$_{CF}$ *Hom*$_{O.C\alpha}$ℭ(−,𝔊−) :
      op-cat ℭ ×$_C$ 𝔇 ↦↦$_{C\alpha}$ *cat-Set* α
      by (*rule NT.is-ntcf-axioms*)
    from *op-op*-Φ **have** *dom-lhs*:
      $\mathcal{D}_\circ$ (*op-cf-adj* (*op-cf-adj* Φ)(|*AdjNT*|)(|*NTMap*|)) = (*op-cat* ℭ ×$_C$ 𝔇)(|*Obj*|)
      by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
    show *op-cf-adj* (*op-cf-adj* Φ)(|*AdjNT*|)(|*NTMap*|) = Φ(|*AdjNT*|)(|*NTMap*|)
    proof(*rule vsv-eqI*, *unfold NT.ntcf-NTMap-vdomain dom-lhs*)
      fix *cd* **assume** *prems*: *cd* ∈$_\circ$ (*op-cat* ℭ ×$_C$ 𝔇)(|*Obj*|)
      then obtain *c d*
        where *cd-def*: *cd* = [*c*, *d*]$_\circ$
          and *c*: *c* ∈$_\circ$ *op-cat* ℭ(|*Obj*|)
          and *d*: *d* ∈$_\circ$ 𝔇(|*Obj*|)
        by (*elim cat-prod-2-ObjE*[*OF L.category-op R.category-axioms prems*])
      from *is-cf-adjunction-axioms c d L.category-axioms R.category-axioms* Φ
      show *op-cf-adj* (*op-cf-adj* Φ)(|*AdjNT*|)(|*NTMap*|)(|*cd*|) = Φ(|*AdjNT*|)(|*NTMap*|)(|*cd*|)
        unfolding *cd-def cat-op-simps*
        by
          (
            cs-concl
              cs-intro:
                cat-arrow-cs-intros
                ntcf-cs-intros
                adj-cs-intros
                cat-op-intros

```
            cat-cs-intros
            cat-prod-cs-intros
        cs-simp: cat-cs-simps cat-op-simps
      )
  qed (auto intro: inv-ntcf-NTMap-vsv)
 qed simp-all
qed (auto intro: adj-cs-intros)

lemmas [cat-op-simps] = is-cf-adjunction.cf-adjunction-op-cf-adj-op-cf-adj
```

13.3.2 Alternative form of the naturality condition

The lemmas in this subsection are based on the comments on page 81 in [9].

```
lemma (in is-cf-adjunction) cf-adj-Comp-commute-RL:
  assumes x ∈∘ ℭ⦇Obj⦈
    and f : 𝔉⦇ObjMap⦈⦇x⦈ ↦𝔇 a
    and k : a ↦𝔇 a'
  shows
    𝔊⦇ArrMap⦈⦇k⦈ ∘A ℭ (Φ⦇AdjNT⦈⦇NTMap⦈⦇x, a⦈∙)⦇ArrVal⦈⦇f⦈ =
      (Φ⦇AdjNT⦈⦇NTMap⦈⦇x, a'⦈∙)⦇ArrVal⦈⦇k ∘A𝔇 f⦈
proof–
  from
    assms
    is-cf-adjunction-axioms
    L.category-axioms R.category-axioms
    L.category-op R.category-op
  have φ-x-a: Φ⦇AdjNT⦈⦇NTMap⦈⦇x, a⦈∙ :
    Hom 𝔇 (𝔉⦇ObjMap⦈⦇x⦈) a ↦cat-Set α Hom ℭ x (𝔊⦇ObjMap⦈⦇a⦈)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
      )
  note φ-x-a-f =
    cat-Set-ArrVal-app-vrange[OF φ-x-a, unfolded in-Hom-iff, OF assms(2)]
  from
    is-cf-adjunction-axioms assms
    L.category-axioms R.category-axioms
    L.category-op R.category-op
  have φ-x-a':
    Φ⦇AdjNT⦈⦇NTMap⦈⦇x, a'⦈∙ :
      Hom 𝔇 (𝔉⦇ObjMap⦈⦇x⦈) a' ↦cat-Set α Hom ℭ x (𝔊⦇ObjMap⦈⦇a'⦈)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
      )
  from is-cf-adjunction-axioms this assms have x-k:
    [ℭ⦇CId⦈⦇x⦈, k]∘ : [x, a]∘ ↦op-cat ℭ ×C 𝔇 [x, a']∘
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
      )
  from
```

NT.ntcf-Comp-commute[*OF this*] *is-cf-adjunction-axioms assms*
L.category-axioms R.category-axioms
L.category-op R.category-op
have
$\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a' \rfloor)_{\bullet} \circ_{A\ cat\text{-}Set\ \alpha}$ *cf-hom* $\mathfrak{D}$ $[\mathfrak{D}(\lfloor CId \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor),\ k]_{\circ}$ =
cf-hom $\mathfrak{C}$ $[\mathfrak{C}(\lfloor CId \rfloor)(\lfloor x \rfloor),\ \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor k \rfloor)]_{\circ} \circ_{A\ cat\text{-}Set\ \alpha} \Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_{\bullet}$
(**is** ‹*?lhs* = *?rhs*›)
by
(
cs-prems **cs-ist-simple**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
moreover from
is-cf-adjunction-axioms assms φ*-x-a'*
L.category-axioms R.category-axioms
L.category-op R.category-op
have *?lhs*$(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$ = $(\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a' \rfloor)_{\bullet})(\lfloor ArrVal \rfloor)(\lfloor k \circ_{A\mathfrak{D}} f \rfloor)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
moreover from
is-cf-adjunction-axioms assms φ*-x-a-f*
L.category-axioms R.category-axioms
L.category-op R.category-op
have
?rhs$(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$ = $\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor k \rfloor) \circ_{A\mathfrak{C}} (\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_{\bullet})(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *is-cf-adjunction*) *cf-adj-Comp-commute-LR*:
assumes $x \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
and $f : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \mapsto_{\mathfrak{D}} a$
and $h : x' \mapsto_{\mathfrak{C}} x$
shows
$(\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_{\bullet})(\lfloor ArrVal \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} h$ =
$(\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x',\ a \rfloor)_{\bullet})(\lfloor ArrVal \rfloor)(\lfloor f \circ_{A\mathfrak{D}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor h \rfloor) \rfloor)$
proof–
from
is-cf-adjunction-axioms assms
L.category-axioms R.category-axioms
L.category-op R.category-op
have φ*-x-a*: $\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_{\bullet}$:
Hom $\mathfrak{D}$ $(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor))$ $a \mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}$ x $(\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*

)
note φ-*x-a-f* =
cat-Set-ArrVal-app-vrange[*OF* φ-*x-a*, *unfolded in-Hom-iff*, *OF assms*(*2*)]
from *is-cf-adjunction-axioms assms* **have**
$[h,\ \mathfrak{D}(\lfloor CId \rfloor)(\lfloor a \rfloor)]_\circ : [x,\ a]_\circ \mapsto_{op\text{-}cat\ \mathfrak{C}\ \times_C\ \mathfrak{D}}\ [x',\ a]_\circ$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
from
NT.ntcf-Comp-commute[*OF this*] *is-cf-adjunction-axioms assms*
L.category-axioms R.category-axioms
L.category-op R.category-op
have
$\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x',\ a \rfloor)_\bullet \circ_{A\ cat\text{-}Set\ \alpha}\ cf\text{-}hom\ \mathfrak{D}\ [\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor h \rfloor),\ \mathfrak{D}(\lfloor CId \rfloor)(\lfloor a \rfloor)]_\circ =$
$cf\text{-}hom\ \mathfrak{C}\ [h,\ \mathfrak{C}(\lfloor CId \rfloor)(\lfloor \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)]_\circ \circ_{A\ cat\text{-}Set\ \alpha}\ \Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_\bullet$
(**is** ‹*?lhs* = *?rhs*›)
by
(
cs-prems
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
moreover from
is-cf-adjunction-axioms assms
L.category-axioms R.category-axioms
L.category-op R.category-op
have *?lhs*$(\lfloor ArrVal \rfloor)(\lfloor f \rfloor) = (\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x',\ a \rfloor)_\bullet)(\lfloor ArrVal \rfloor)(\lfloor f \circ_{A\mathfrak{D}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor h \rfloor) \rfloor)$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
moreover from
is-cf-adjunction-axioms assms φ-*x-a-f*
L.category-axioms R.category-axioms
L.category-op R.category-op
have *?rhs*$(\lfloor ArrVal \rfloor)(\lfloor f \rfloor) = (\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_\bullet)(\lfloor ArrVal \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} h$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
ultimately show *?thesis* **by** *simp*
qed

13.4 Unit

13.4.1 Definition and elementary properties

See Chapter IV-1 in [9].

definition *cf-adjunction-unit* :: $V \Rightarrow V$ ($\langle \eta_C \rangle$)
where $\eta_C\ \Phi$ =
[

```
    (
      λx∈₀Φ(|AdjLeft|)(|HomDom|)(|Obj|).
        (Φ(|AdjNT|)(|NTMap|)(|x, Φ(|AdjLeft|)(|ObjMap|)(|x|)|)•)(|ArrVal|)(|
          Φ(|AdjLeft|)(|HomCod|)(|CId|)(Φ(|AdjLeft|)(|ObjMap|)(|x|))
        |)
    ),
    cf-id (Φ(|AdjLeft|)(|HomDom|)),
    (Φ(|AdjRight|)) ∘CF (Φ(|AdjLeft|)),
    Φ(|AdjLeft|)(|HomDom|),
    Φ(|AdjLeft|)(|HomDom|)
  ]₀
```

Components.

lemma *cf-adjunction-unit-components*:
  shows $\eta_C\ \Phi(|NTMap|) =$
    (
      $\lambda x \in_\circ \Phi(|AdjLeft|)(|HomDom|)(|Obj|).$
        $(\Phi(|AdjNT|)(|NTMap|)(|x, \Phi(|AdjLeft|)(|ObjMap|)(|x|)|)_\bullet)(|ArrVal|)(|$
          $\Phi(|AdjLeft|)(|HomCod|)(|CId|)(\Phi(|AdjLeft|)(|ObjMap|)(|x|))$
        $|)$
)
    and $\eta_C\ \Phi(|NTDom|) =$ *cf-id* $(\Phi(|AdjLeft|)(|HomDom|))$
    and $\eta_C\ \Phi(|NTCod|) = (\Phi(|AdjRight|)) \circ_{CF} (\Phi(|AdjLeft|))$
    and $\eta_C\ \Phi(|NTDGDom|) = \Phi(|AdjLeft|)(|HomDom|)$
    and $\eta_C\ \Phi(|NTDGCod|) = \Phi(|AdjLeft|)(|HomDom|)$
  unfolding *cf-adjunction-unit-def nt-field-simps*
  by (*simp-all add*: *nat-omega-simps*)

context *is-cf-adjunction*
begin

lemma *cf-adjunction-unit-components′*:
  shows $\eta_C\ \Phi(|NTMap|) =$
    $(\lambda x \in_\circ \mathfrak{C}(|Obj|).\ (\Phi(|AdjNT|)(|NTMap|)(|x, \mathfrak{F}(|ObjMap|)(|x|)|)_\bullet)(|ArrVal|)(|\mathfrak{D}(|CId|)(|\mathfrak{F}(|ObjMap|)(|x|)|)|))$
    and $\eta_C\ \Phi(|NTDom|) =$ *cf-id* $\mathfrak{C}$
    and $\eta_C\ \Phi(|NTCod|) = \mathfrak{G} \circ_{CF} \mathfrak{F}$
    and $\eta_C\ \Phi(|NTDGDom|) = \mathfrak{C}$
    and $\eta_C\ \Phi(|NTDGCod|) = \mathfrak{C}$
  unfolding *cf-adjunction-unit-components*
  by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps adj-cs-simps*)+

mk-VLambda *cf-adjunction-unit-components′*(*1*)
  |*vdomain cf-adjunction-unit-NTMap-vdomain*[*adj-cs-simps*]|
  |*app cf-adjunction-unit-NTMap-app*[*adj-cs-simps*]|

end

mk-VLambda *cf-adjunction-unit-components*(*1*)
  |*vsv cf-adjunction-unit-NTMap-vsv*[*adj-cs-intros*]|

lemmas [*adj-cs-simps*] =
  is-cf-adjunction.cf-adjunction-unit-NTMap-vdomain
  is-cf-adjunction.cf-adjunction-unit-NTMap-app

13.4.2 Natural transformation map

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-is-arr*:
  assumes $x \in_\circ \mathfrak{C}(|Obj|)$

shows $\eta_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor x \rfloor) : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor)$
proof–
from
is-cf-adjunction-axioms assms
L.category-axioms R.category-axioms
L.category-op R.category-op
have φ-x-$\mathfrak{F}x$:
$\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x, \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor)_{\bullet}$:
Hom $\mathfrak{D}$ $(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor))$ $(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor)) \mapsto_{cat\text{-}Set\ \alpha}$
Hom $\mathfrak{C}$ x $(\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor))$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
)
from *is-cf-adjunction-axioms assms* **have** *CId*-$\mathfrak{F}x$:
$\mathfrak{D}(\lfloor CId \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \mapsto_{\mathfrak{D}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor)$
by (*cs-concl* **cs-intro**: *cat-cs-intros adj-cs-intros*)
from
is-cf-adjunction-axioms
assms
cat-Set-ArrVal-app-vrange[*OF* φ-x-$\mathfrak{F}x$, *unfolded in-Hom-iff*, *OF CId*-$\mathfrak{F}x$]
show $\eta_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor x \rfloor) : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor)$
by (*cs-concl* **cs-shallow cs-simp**: *adj-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-is-arr′*:
assumes $x \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
and $a = x$
and $b = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor)$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\eta_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor x \rfloor) : x \mapsto_{\mathfrak{C}'} b$
using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cf-adjunction-unit-NTMap-is-arr*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-unit-NTMap-is-arr′*

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-vrange*:
$\mathcal{R}_{\circ}\ (\eta_C\ \Phi(\lfloor NTMap \rfloor)) \subseteq_{\circ} \mathfrak{C}(\lfloor Arr \rfloor)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-adjunction-unit-NTMap-vdomain*)
fix x **assume** *prems*: $x \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
from *cf-adjunction-unit-NTMap-is-arr*[*OF prems*] **show** $\eta_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor x \rfloor) \in_{\circ} \mathfrak{C}(\lfloor Arr \rfloor)$
by *auto*
qed (*auto intro*: *adj-cs-intros*)

13.4.3 Unit is a natural transformation

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-is-ntcf*:
$\eta_C\ \Phi : \mathit{cf\text{-}id}\ \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-ntcfI′*)
show *vfsequence* $(\eta_C\ \Phi)$ **unfolding** *cf-adjunction-unit-def* **by** *simp*
show *vcard* $(\eta_C\ \Phi) = 5_{\mathbb{N}}$
unfolding *cf-adjunction-unit-def* **by** (*simp add*: *nat-omega-simps*)
from *is-cf-adjunction-axioms* **show** *cf-id* $\mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros adj-cs-intros*)
from *is-cf-adjunction-axioms* **show** $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros adj-cs-intros*)
from *is-cf-adjunction-axioms* **show** $\mathcal{D}_{\circ}\ (\eta_C\ \Phi(\lfloor NTMap \rfloor)) = \mathfrak{C}(\lfloor Obj \rfloor)$

```
    by (cs-concl cs-shallow cs-simp: adj-cs-simps cs-intro: cat-cs-intros)
  show η_C Φ(|NTMap|)(|a|) : cf-id 𝔈(|ObjMap|)(|a|) ↦_𝔈 (𝔊 ∘_CF 𝔉)(|ObjMap|)(|a|)
    if a ∈_∘ 𝔈(|Obj|) for a
    using is-cf-adjunction-axioms that
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)
  show
    η_C Φ(|NTMap|)(|b|) ∘_A𝔈 cf-id 𝔈(|ArrMap|)(|f|) =
      (𝔊 ∘_CF 𝔉)(|ArrMap|)(|f|) ∘_A𝔈 η_C Φ(|NTMap|)(|a|)
    if f : a ↦_𝔈 b for a b f
    using is-cf-adjunction-axioms that
    by
      (
        cs-concl
          cs-simp:
            cf-adj-Comp-commute-RL cf-adj-Comp-commute-LR
            cat-cs-simps
            adj-cs-simps
          cs-intro: cat-cs-intros adj-cs-intros
      )
qed (auto simp: cf-adjunction-unit-components′)

lemma (in is-cf-adjunction) cf-adjunction-unit-is-ntcf′:
  assumes 𝔖 = cf-id 𝔈
    and 𝔖′ = 𝔊 ∘_CF 𝔉
    and 𝔄 = 𝔈
    and 𝔅 = 𝔈
  shows η_C Φ : 𝔖 ↦_CF 𝔖′ : 𝔄 ↦↦_Cα 𝔅
  unfolding assms by (rule cf-adjunction-unit-is-ntcf)

lemmas [adj-cs-intros] = is-cf-adjunction.cf-adjunction-unit-is-ntcf′
```

13.4.4 Every component of a unit is a universal arrow

The lemmas in this subsection are based on elements of the statement of Theorem 1 in Chapter IV-1 in [9].

```
lemma (in is-cf-adjunction) cf-adj-umap-of-unit:
  assumes x ∈_∘ 𝔈(|Obj|) and a ∈_∘ 𝔇(|Obj|)
  shows Φ(|AdjNT|)(|NTMap|)(|x, a|)_• = umap-of 𝔊 x (𝔉(|ObjMap|)(|x|)) (η_C Φ(|NTMap|)(|x|)) a
  (is ‹Φ(|AdjNT|)(|NTMap|)(|x, a|)_• = ?uof-a›)
proof–

  from
    is-cf-adjunction-axioms assms
    L.category-axioms R.category-axioms
    L.category-op R.category-op
  have φ-xa: Φ(|AdjNT|)(|NTMap|)(|x, a|)_• :
    Hom 𝔇 (𝔉(|ObjMap|)(|x|)) a ↦_cat-Set α Hom 𝔈 x (𝔊(|ObjMap|)(|a|))
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
      )
  then have dom-lhs:
    𝒟_∘ ((Φ(|AdjNT|)(|NTMap|)(|x, a|)_•)(|ArrVal|)) = Hom 𝔇 (𝔉(|ObjMap|)(|x|)) a
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from is-cf-adjunction-axioms assms have uof-a:
```

```
    ?uof-a : Hom 𝔇 (ℱ(|ObjMap|)(|x|)) a ↦_cat-Set α Hom ℭ x (𝔊(|ObjMap|)(|a|))
    by (cs-concl cs-intro: cat-cs-intros adj-cs-intros)
  then have dom-rhs: 𝒟∘ (?uof-a(|ArrVal|)) = Hom 𝔇 (ℱ(|ObjMap|)(|x|)) a
    by (cs-concl cs-simp: cat-cs-simps)

  show ?thesis
  proof(rule arr-Set-eqI[of α])
    from φ-xa show arr-Set-φ-xa: arr-Set α (Φ(|AdjNT|)(|NTMap|)(|x, a|)•)
      by (auto dest: cat-Set-is-arrD(1))
    from uof-a show arr-Set-uof-a: arr-Set α ?uof-a
      by (auto dest: cat-Set-is-arrD(1))
    show (Φ(|AdjNT|)(|NTMap|)(|x, a|)•)(|ArrVal|) = ?uof-a(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix g assume prems: g : ℱ(|ObjMap|)(|x|) ↦_𝔇 a
      from is-cf-adjunction-axioms assms prems show
        (Φ(|AdjNT|)(|NTMap|)(|x, a|)•)(|ArrVal|)(|g|) = ?uof-a(|ArrVal|)(|g|)
        by
          (
            cs-concl cs-shallow
              cs-simp:
                cf-adj-Comp-commute-RL
                adj-cs-simps
                cat-cs-simps
                cat-op-simps
                cat-prod-cs-simps
              cs-intro:
                adj-cs-intros
                ntcf-cs-intros
                cat-cs-intros
                cat-op-intros
                cat-prod-cs-intros
          )
    qed (use arr-Set-φ-xa arr-Set-uof-a in auto)

  qed (use φ-xa uof-a in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed

lemma (in is-cf-adjunction) cf-adj-umap-of-unit':
  assumes x ∈∘ ℭ(|Obj|)
    and a ∈∘ 𝔇(|Obj|)
    and η = η_C Φ(|NTMap|)(|x|)
    and 𝔉x = ℱ(|ObjMap|)(|x|)
  shows Φ(|AdjNT|)(|NTMap|)(|x, a|)• = umap-of 𝔊 x 𝔉x η a
  using assms(1,2) unfolding assms(3,4) by (rule cf-adj-umap-of-unit)

lemma (in is-cf-adjunction) cf-adjunction-unit-component-is-ua-of:
  assumes x ∈∘ ℭ(|Obj|)
  shows universal-arrow-of 𝔊 x (ℱ(|ObjMap|)(|x|)) (η_C Φ(|NTMap|)(|x|))
    (is ‹universal-arrow-of 𝔊 x (ℱ(|ObjMap|)(|x|)) ?ηx›)
proof(rule RL.cf-ua-of-if-ntcf-ua-of-is-iso-ntcf)
  from is-cf-adjunction-axioms assms show ℱ(|ObjMap|)(|x|) ∈∘ 𝔇(|Obj|)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros adj-cs-intros)
  from is-cf-adjunction-axioms assms show
    η_C Φ(|NTMap|)(|x|) : x ↦_ℭ 𝔊(|ObjMap|)(ℱ(|ObjMap|)(|x|))
    by (cs-concl cs-shallow cs-intro: cat-cs-intros adj-cs-intros)
  show
    ntcf-ua-of α 𝔊 x (ℱ(|ObjMap|)(|x|)) (η_C Φ(|NTMap|)(|x|)) :
```

```
    Hom_{O.Cα}𝔇(𝔉(|ObjMap|)(|x|),−) ↦_{CF.iso} Hom_{O.Cα}ℭ(x,−) ∘_{CF} 𝔊 :
    𝔇 ↦↦_{Cα} cat-Set α
  (is ‹?ntcf-ua-of : ?H𝔉 ↦_{CF.iso} ?H𝔊 : 𝔇 ↦↦_{Cα} cat-Set α›)
 proof(rule is-iso-ntcfI)
  from is-cf-adjunction-axioms assms show
    ?ntcf-ua-of : ?H𝔉 ↦_{CF} ?H𝔊 : 𝔇 ↦↦_{Cα} cat-Set α
    by (intro RL.cf-ntcf-ua-of-is-ntcf)
      (cs-concl cs-shallow cs-intro: cat-cs-intros adj-cs-intros)+
  fix a assume prems: a ∈_∘ 𝔇(|Obj|)
  from assms prems have
    Φ(|AdjNT|)(|NTMap|)(|x, a|)_• = umap-of 𝔊 x (𝔉(|ObjMap|)(|x|)) ?ηx a
    (is ‹Φ(|AdjNT|)(|NTMap|)(|x, a|)_• = ?uof-a›)
    by (rule cf-adj-umap-of-unit)
  from assms prems L.category-axioms R.category-axioms have
    [x, a]_∘ ∈_∘ (op-cat ℭ ×_C 𝔇)(|Obj|)
    by (cs-concl cs-shallow cs-intro: cat-op-intros cat-prod-cs-intros)
  from
    NT.iso-ntcf-is-iso-arr[
      OF this, unfolded cf-adj-umap-of-unit[OF assms prems]
      ]
    is-cf-adjunction-axioms assms prems
    L.category-axioms R.category-axioms
  have ?uof-a : Hom 𝔇 (𝔉(|ObjMap|)(|x|)) a ↦_{iso cat-Set α} Hom ℭ x (𝔊(|ObjMap|)(|a|))
    by
      (
        cs-prems
          cs-simp: cat-cs-simps
          cs-intro:
            cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
      )
  with is-cf-adjunction-axioms assms prems show
    ?ntcf-ua-of(|NTMap|)(|a|) : ?H𝔉(|ObjMap|)(|a|) ↦_{iso cat-Set α} ?H𝔊(|ObjMap|)(|a|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros adj-cs-intros
      )
 qed
qed
```

13.5 Counit

13.5.1 Definition and elementary properties

```
definition cf-adjunction-counit :: V ⇒ V (‹ε_C›)
  where ε_C Φ =
    [
      (
        λx∈_∘Φ(|AdjLeft|)(|HomCod|)(|Obj|).
          (Φ(|AdjNT|)(|NTMap|)(Φ(|AdjRight|)(|ObjMap|)(|x|), x|)_•)^{-1}_{Set}(|ArrVal|)(|
            Φ(|AdjLeft|)(|HomDom|)(|CId|)(Φ(|AdjRight|)(|ObjMap|)(|x|))
            |)
      ),
      (Φ(|AdjLeft|)) ∘_{CF} (Φ(|AdjRight|)),
      cf-id (Φ(|AdjLeft|)(|HomCod|)),
      Φ(|AdjLeft|)(|HomCod|),
      Φ(|AdjLeft|)(|HomCod|)
```

]$_\circ$

Components.

lemma *cf-adjunction-counit-components*:
shows $\varepsilon_C\ \Phi(\lfloor NTMap \rfloor) =$
(
$\lambda x \in_\circ \Phi(\lfloor AdjLeft \rfloor)(\lfloor HomCod \rfloor)(\lfloor Obj \rfloor).$
$(\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\Phi(\lfloor AdjRight \rfloor)(\lfloor ObjMap \rfloor)(\lfloor x \rfloor),\ x)_\bullet)^{-1}{}_{Set}(\lfloor ArrVal \rfloor)(\lfloor$
$\Phi(\lfloor AdjLeft \rfloor)(\lfloor HomDom \rfloor)(\lfloor CId \rfloor)(\Phi(\lfloor AdjRight \rfloor)(\lfloor ObjMap \rfloor)(\lfloor x \rfloor))$
$\rfloor)$
)
and $\varepsilon_C\ \Phi(\lfloor NTDom \rfloor) = (\Phi(\lfloor AdjLeft \rfloor)) \circ_{CF} (\Phi(\lfloor AdjRight \rfloor))$
and $\varepsilon_C\ \Phi(\lfloor NTCod \rfloor) = cf\text{-}id\ (\Phi(\lfloor AdjLeft \rfloor)(\lfloor HomCod \rfloor))$
and $\varepsilon_C\ \Phi(\lfloor NTDGDom \rfloor) = \Phi(\lfloor AdjLeft \rfloor)(\lfloor HomCod \rfloor)$
and $\varepsilon_C\ \Phi(\lfloor NTDGCod \rfloor) = \Phi(\lfloor AdjLeft \rfloor)(\lfloor HomCod \rfloor)$
unfolding *cf-adjunction-counit-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

context *is-cf-adjunction*
begin

lemma *cf-adjunction-counit-components′*:
shows $\varepsilon_C\ \Phi(\lfloor NTMap \rfloor) =$
(
$\lambda x \in_\circ \mathfrak{D}(\lfloor Obj \rfloor).$
$(\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor),\ x)_\bullet)^{-1}{}_{Set}(\lfloor ArrVal \rfloor)(\lfloor \mathfrak{C}(\lfloor CId \rfloor)(\lfloor \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor) \rfloor)$
)
and $\varepsilon_C\ \Phi(\lfloor NTDom \rfloor) = \mathfrak{F} \circ_{CF} \mathfrak{G}$
and $\varepsilon_C\ \Phi(\lfloor NTCod \rfloor) = cf\text{-}id\ \mathfrak{D}$
and $\varepsilon_C\ \Phi(\lfloor NTDGDom \rfloor) = \mathfrak{D}$
and $\varepsilon_C\ \Phi(\lfloor NTDGCod \rfloor) = \mathfrak{D}$
unfolding *cf-adjunction-counit-components*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps adj-cs-simps*)+

mk-VLambda *cf-adjunction-counit-components′*(*1*)
|*vdomain cf-adjunction-counit-NTMap-vdomain*[*adj-cs-simps*]|
|*app cf-adjunction-counit-NTMap-app*[*adj-cs-simps*]|

end

mk-VLambda *cf-adjunction-counit-components*(*1*)
|*vsv cf-adjunction-counit-NTMap-vsv*[*adj-cs-intros*]|

lemmas [*adj-cs-simps*] =
is-cf-adjunction.cf-adjunction-counit-NTMap-vdomain
is-cf-adjunction.cf-adjunction-counit-NTMap-app

13.5.2 Duality for the unit and counit

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-op*:
$\eta_C\ (op\text{-}cf\text{-}adj\ \Phi)(\lfloor NTMap \rfloor) = \varepsilon_C\ \Phi(\lfloor NTMap \rfloor)$
proof–
interpret *op-Φ*:
is-cf-adjunction α ‹*op-cat* $\mathfrak{D}$› ‹*op-cat* $\mathfrak{C}$› ‹*op-cf* $\mathfrak{G}$› ‹*op-cf* $\mathfrak{F}$› ‹*op-cf-adj* Φ›
by (*rule is-cf-adjunction-op*)
show *?thesis*
proof
(

```
    rule vsv-eqI,
    unfold
      cf-adjunction-counit-NTMap-vdomain
      op-Φ.cf-adjunction-unit-NTMap-vdomain
  )
  fix a assume prems: a ∈∘ op-cat 𝔇(|Obj|)
  then have a: a ∈∘ 𝔇(|Obj|) unfolding cat-op-simps by simp
  from is-cf-adjunction-axioms a show
    η_C (op-cf-adj Φ)(|NTMap|)(|a|) = ε_C Φ(|NTMap|)(|a|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Set-cs-simps cat-cs-simps cat-op-simps adj-cs-simps
          cs-intro:
            cat-arrow-cs-intros cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed
  (
    simp-all add:
      cat-op-simps cf-adjunction-counit-NTMap-vsv cf-adjunction-unit-NTMap-vsv
  )
qed


lemmas [cat-op-simps] = is-cf-adjunction.cf-adjunction-unit-NTMap-op

lemma (in is-cf-adjunction) cf-adjunction-counit-NTMap-op:
  ε_C (op-cf-adj Φ)(|NTMap|) = η_C Φ(|NTMap|)
  by
    (
      rule is-cf-adjunction.cf-adjunction-unit-NTMap-op[
        OF is-cf-adjunction-op,
        unfolded is-cf-adjunction.cf-adjunction-op-cf-adj-op-cf-adj[
          OF is-cf-adjunction-axioms
          ],
        unfolded cat-op-simps,
        symmetric
      ]
   )

lemmas [cat-op-simps] = is-cf-adjunction.cf-adjunction-counit-NTMap-op

lemma (in is-cf-adjunction) op-ntcf-cf-adjunction-counit:
  op-ntcf (ε_C Φ) = η_C (op-cf-adj Φ)
  (is ‹?ε = ?η›)
proof(rule vsv-eqI)
  interpret op-Φ:
    is-cf-adjunction α ‹op-cat 𝔇› ‹op-cat ℭ› ‹op-cf 𝔊› ‹op-cf 𝔉› ‹op-cf-adj Φ›
    by (rule is-cf-adjunction-op)
  have dom-lhs: 𝒟∘ ?ε = 5_ℕ unfolding op-ntcf-def by (simp add: nat-omega-simps)
  have dom-rhs: 𝒟∘ ?η = 5_ℕ
    unfolding cf-adjunction-unit-def by (simp add: nat-omega-simps)
  show 𝒟∘ ?ε = 𝒟∘ ?η unfolding dom-lhs dom-rhs by simp
  show a ∈∘ 𝒟∘ ?ε ⟹ ?ε(|a|) = ?η(|a|) for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
```

fold nt-field-simps,
unfold cf-adjunction-unit-NTMap-op,
unfold
cf-adjunction-counit-components′
cf-adjunction-unit-components′
op-Φ.cf-adjunction-counit-components′
op-Φ.cf-adjunction-unit-components′
cat-op-simps
)
simp-all
qed (*auto simp*: *op-ntcf-def cf-adjunction-unit-def*)

lemmas [*cat-op-simps*] = *is-cf-adjunction.op-ntcf-cf-adjunction-counit*

lemma (**in** *is-cf-adjunction*) *op-ntcf-cf-adjunction-unit*:
op-ntcf (η_C Φ) = ε_C (*op-cf-adj* Φ)
(**is** ‹*?η* = *?ε*›)
proof(*rule vsv-eqI*)
interpret *op-Φ*:
is-cf-adjunction α ‹*op-cat* 𝔇› ‹*op-cat* ℭ› ‹*op-cf* 𝔊› ‹*op-cf* 𝔉› ‹*op-cf-adj* Φ›
by (*rule is-cf-adjunction-op*)
have *dom-lhs*: $\mathcal{D}_\circ$ *?η* = $5_{\mathbb{N}}$
unfolding *op-ntcf-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ *?ε* = $5_{\mathbb{N}}$
unfolding *cf-adjunction-counit-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ *?η* = $\mathcal{D}_\circ$ *?ε* **unfolding** *dom-lhs dom-rhs* **by** *simp*
show *a* $\in_\circ$ $\mathcal{D}_\circ$ *?η* $\Longrightarrow$ *?η*(⁅*a*⁆) = *?ε*(⁅*a*⁆) **for** *a*
by
(
unfold dom-lhs,
elim-in-numeral,
fold nt-field-simps,
unfold cf-adjunction-counit-NTMap-op,
unfold
cf-adjunction-counit-components′
cf-adjunction-unit-components′
op-Φ.cf-adjunction-counit-components′
op-Φ.cf-adjunction-unit-components′
cat-op-simps
)
simp-all
qed (*auto simp*: *op-ntcf-def cf-adjunction-counit-def*)

lemmas [*cat-op-simps*] = *is-cf-adjunction.op-ntcf-cf-adjunction-unit*

13.5.3 Natural transformation map

lemma (**in** *is-cf-adjunction*) *cf-adjunction-counit-NTMap-is-arr*:
assumes *x* $\in_\circ$ 𝔇(⁅*Obj*⁆)
shows ε_C Φ(⁅*NTMap*⁆)(⁅*x*⁆) : 𝔉(⁅*ObjMap*⁆)(⁅𝔊(⁅*ObjMap*⁆)(⁅*x*⁆)⁆) $\mapsto_{\mathfrak{D}}$ *x*
proof–
from *assms* **have** *x*: *x* $\in_\circ$ *op-cat* 𝔇(⁅*Obj*⁆) **unfolding** *cat-op-simps* **by** *simp*
show *?thesis*
by
(
rule is-cf-adjunction.cf-adjunction-unit-NTMap-is-arr[
OF is-cf-adjunction-op x,
unfolded cf-adjunction-unit-NTMap-op cat-op-simps

]
)
qed

lemma (**in** *is-cf-adjunction*) *cf-adjunction-counit-NTMap-is-arr'*:
assumes $x \in_{\circ} \mathfrak{D}(\lfloor Obj \rfloor)$
and $a = \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \rfloor)$
and $b = x$
and $\mathfrak{D}' = \mathfrak{D}$
shows $\varepsilon_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor x \rfloor) : a \mapsto_{\mathfrak{D}'} b$
using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cf-adjunction-counit-NTMap-is-arr*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-counit-NTMap-is-arr'*

lemma (**in** *is-cf-adjunction*) *cf-adjunction-counit-NTMap-vrange*:
$\mathcal{R}_{\circ}\ (\varepsilon_C\ \Phi(\lfloor NTMap \rfloor)) \subseteq_{\circ} \mathfrak{D}(\lfloor Arr \rfloor)$
by
(
rule is-cf-adjunction.cf-adjunction-unit-NTMap-vrange[
OF is-cf-adjunction-op,
unfolded cf-adjunction-unit-NTMap-op cat-op-simps
]
)

13.5.4 Counit is a natural transformation

lemma (**in** *is-cf-adjunction*) *cf-adjunction-counit-is-ntcf*:
$\varepsilon_C\ \Phi : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \textit{cf-id}\ \mathfrak{D} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof–
from *is-cf-adjunction.cf-adjunction-unit-is-ntcf*[*OF is-cf-adjunction-op*] **have**
$\varepsilon_C\ \Phi :$
op-cf (*op-cf* $\mathfrak{F} \circ_{CF}$ *op-cf* $\mathfrak{G}$) $\mapsto_{CF}$ *op-cf* (*cf-id* (*op-cat* $\mathfrak{D}$)) :
op-cat (*op-cat* $\mathfrak{D}$) $\mapsto\mapsto_{C\alpha}$ *op-cat* (*op-cat* $\mathfrak{D}$)
unfolding
is-cf-adjunction.op-ntcf-cf-adjunction-unit[
OF is-cf-adjunction-op, *unfolded cat-op-simps*, *symmetric*
]
by (*rule is-ntcf.is-ntcf-op*)
then show *?thesis* **unfolding** *cat-op-simps* **.**
qed

lemma (**in** *is-cf-adjunction*) *cf-adjunction-counit-is-ntcf'*:
assumes $\mathfrak{S} = \mathfrak{F} \circ_{CF} \mathfrak{G}$
and $\mathfrak{S}' = \textit{cf-id}\ \mathfrak{D}$
and $\mathfrak{A} = \mathfrak{D}$
and $\mathfrak{B} = \mathfrak{D}$
shows $\varepsilon_C\ \Phi : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule cf-adjunction-counit-is-ntcf*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-counit-is-ntcf'*

13.5.5 Every component of a counit is a universal arrow

The lemmas in this subsection are based on elements of the statement of Theorem 1 in Chapter IV-1 in [9].

lemma (**in** *is-cf-adjunction*) *cf-adj-umap-fo-counit*:
assumes $x \in_{\circ} \mathfrak{D}(\lfloor Obj \rfloor)$ **and** $a \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
shows *op-cf-adj* $\Phi(\lfloor AdjNT \rfloor)(\lfloor NTMap \rfloor)(\lfloor x,\ a \rfloor)_{\bullet}$ =

umap-fo $\mathfrak{F}$ x $(\mathfrak{G}(\!|ObjMap|\!)(\!|x|\!))$ $(\varepsilon_C\ \Phi(\!|NTMap|\!)(\!|x|\!))$ a
by
(
rule is-cf-adjunction.cf-adj-umap-of-unit[
OF is-cf-adjunction-op,
unfolded cat-op-simps,
OF assms,
unfolded cf-adjunction-unit-NTMap-op
]
)

lemma (**in** *is-cf-adjunction*) *cf-adjunction-counit-component-is-ua-fo*:
assumes $x \in_\circ \mathfrak{D}(\!|Obj|\!)$
shows *universal-arrow-fo* $\mathfrak{F}$ x $(\mathfrak{G}(\!|ObjMap|\!)(\!|x|\!))$ $(\varepsilon_C\ \Phi(\!|NTMap|\!)(\!|x|\!))$
by
(
rule is-cf-adjunction.cf-adjunction-unit-component-is-ua-of[
OF is-cf-adjunction-op,
unfolded cat-op-simps,
OF assms,
unfolded cf-adjunction-unit-NTMap-op
]
)

13.5.6 Further properties

lemma (**in** *is-cf-adjunction*) *cf-adj-AdjNT-cf-adjunction-unit*:
— See Chapter IV-1 in [9].
assumes $x \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $f : \mathfrak{F}(\!|ObjMap|\!)(\!|x|\!) \mapsto_{\mathfrak{D}} a$
shows
$\mathfrak{G}(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{C}} \eta_C\ \Phi(\!|NTMap|\!)(\!|x|\!) =$
$(\Phi(\!|AdjNT|\!)(\!|NTMap|\!)(\!|x,\ a|\!)_\bullet)(\!|ArrVal|\!)(\!|f|\!)$
proof–
from *assms*(*1*) **have** $\mathfrak{D}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|x|\!)|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|x|\!) \mapsto_{\mathfrak{D}} \mathfrak{F}(\!|ObjMap|\!)(\!|x|\!)$
by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *cf-adj-Comp-commute-RL*[*OF assms*(*1*) *this assms*(*2*)] *assms* **show** *?thesis*
by
(
cs-prems **cs-shallow**
cs-simp:
cat-cs-simps
is-cf-adjunction.cf-adjunction-unit-NTMap-app[*symmetric*]
cs-intro: *adj-cs-intros*
)
qed

lemma (**in** *is-cf-adjunction*) *cf-adj-AdjNT-cf-adjunction-counit*:
— See Chapter IV-1 in [9].
assumes $x \in_\circ \mathfrak{D}(\!|Obj|\!)$ **and** $g : a \mapsto_{\mathfrak{C}} \mathfrak{G}(\!|ObjMap|\!)(\!|x|\!)$
shows
$\varepsilon_C\ \Phi(\!|NTMap|\!)(\!|x|\!) \circ_{A\mathfrak{D}} \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) =$
$(\Phi(\!|AdjNT|\!)(\!|NTMap|\!)(\!|a,\ x|\!)_\bullet)^{-1}{}_{C\,cat\text{-}Set}\ \alpha(\!|ArrVal|\!)(\!|g|\!)$
using
is-cf-adjunction.cf-adj-AdjNT-cf-adjunction-unit
[
OF is-cf-adjunction-op,
unfolded cat-op-simps cf-adjunction-unit-NTMap-op,
OF assms

```
    ]
  assms
by
  (
    cs-prems
      cs-simp: cat-cs-simps cat-op-simps
      cs-intro:
        cat-cs-intros
        adj-cs-intros
        cat-op-intros
        cat-prod-cs-intros
  )

lemma (in is-cf-adjunction) cf-adj-counit-unit-app[adj-cs-simps]:
  — See Chapter IV-1 in [9].
  assumes x ∈∘ 𝔇(|Obj|) and g : a ↦𝔈 𝔊(|ObjMap|)(|x|)
  shows 𝔊(|ArrMap|)(|εC Φ(|NTMap|)(|x|) ∘A𝔇 𝔉(|ArrMap|)(|g|)|) ∘A𝔈 ηC Φ(|NTMap|)(|a|) = g
proof–
  from assms(2) have a: a ∈∘ 𝔈(|Obj|) by auto
  from assms have inv-Φ-g:
    (Φ(|AdjNT|)(|NTMap|)(|a, x|)•)⁻¹C cat-Set α(|ArrVal|)(|g|) : 𝔉(|ObjMap|)(|a|) ↦𝔇 x
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro:
            cat-arrow-cs-intros
            cat-cs-intros
            adj-cs-intros
            cat-prod-cs-intros
            cat-op-intros
      )
  from assms show ?thesis
    unfolding
      cf-adj-AdjNT-cf-adjunction-counit[OF assms]
      cf-adj-AdjNT-cf-adjunction-unit[OF a inv-Φ-g]
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro:
            cat-arrow-cs-intros
            cat-cs-intros
            adj-cs-intros
            cat-prod-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-cf-adjunction.cf-adj-counit-unit-app

lemma (in is-cf-adjunction) cf-adj-unit-counit-app[adj-cs-simps]:
  — See Chapter IV-1 in [9].
  assumes x ∈∘ 𝔈(|Obj|) and f : 𝔉(|ObjMap|)(|x|) ↦𝔇 a
  shows εC Φ(|NTMap|)(|a|) ∘A𝔇 𝔉(|ArrMap|)(𝔊(|ArrMap|)(|f|) ∘A𝔈 ηC Φ(|NTMap|)(|x|)) = f
proof–
  from assms(2) have a: a ∈∘ 𝔇(|Obj|) by auto
  from assms have Φ-f:
```

```
    (Φ⦇AdjNT⦈⦇NTMap⦈⦇x, a⦈•)⦇ArrVal⦈⦇f⦈ : x ↦_ℭ 𝔊⦇ObjMap⦈⦇a⦈
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro:
            cat-arrow-cs-intros
            cat-cs-intros
            adj-cs-intros
            cat-prod-cs-intros
            cat-op-intros
      )
  from assms show ?thesis
    unfolding
      cf-adj-AdjNT-cf-adjunction-unit[OF assms]
      cf-adj-AdjNT-cf-adjunction-counit[OF a Φ-f]
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro:
            cat-arrow-cs-intros
            cat-cs-intros
            adj-cs-intros
            cat-prod-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-cf-adjunction.cf-adj-unit-counit-app
```

13.6 Counit-unit equations

The following equations appear as part of the statement of Theorem 1 in Chapter IV-1 in [9]. These equations also appear in [2], where they are named *counit–unit equations*.

```
lemma (in is-cf-adjunction) cf-adjunction-counit-unit:
  (𝔊 ∘_CF−NTCF ε_C Φ) ⋆_NTCF (η_C Φ ∘_NTCF−CF 𝔊) = ntcf-id 𝔊
  (is ‹(𝔊 ∘_CF−NTCF ?ε) ⋆_NTCF (?η ∘_NTCF−CF 𝔊) = ntcf-id 𝔊›)
proof(rule ntcf-eqI)
  from is-cf-adjunction-axioms show
    (𝔊 ∘_CF−NTCF ?ε) ⋆_NTCF (?η ∘_NTCF−CF 𝔊) : 𝔊 ↦_CF 𝔊 : 𝔇 ↦↦_Cα ℭ
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)
  show ntcf-id 𝔊 : 𝔊 ↦_CF 𝔊 : 𝔇 ↦↦_Cα ℭ
    by (rule is-functor.cf-ntcf-id-is-ntcf[OF RL.is-functor-axioms])
  from is-cf-adjunction-axioms have dom-lhs:
    𝒟∘ (((𝔊 ∘_CF−NTCF ?ε) ⋆_NTCF (?η ∘_NTCF−CF 𝔊))⦇NTMap⦈) = 𝔇⦇Obj⦈
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros
      )
  from is-cf-adjunction-axioms have dom-rhs: 𝒟∘ (ntcf-id 𝔊⦇NTMap⦈) = 𝔇⦇Obj⦈
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: adj-cs-intros)
  show ((𝔊 ∘_CF−NTCF ?ε) ⋆_NTCF (?η ∘_NTCF−CF 𝔊))⦇NTMap⦈ = ntcf-id 𝔊⦇NTMap⦈
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume prems: a ∈∘ 𝔇⦇Obj⦈
    let ?φ-aa = ‹Φ⦇AdjNT⦈⦇NTMap⦈⦇𝔊⦇ObjMap⦈⦇a⦈, a⦈•›
```

have *category* α (*cat-Set* α)
  by (*rule category-cat-Set*)
from *is-cf-adjunction-axioms prems*
  L.category-axioms R.category-axioms
  L.category-op R.category-op
  LR.is-functor-axioms RL.is-functor-axioms
  category-cat-Set
have
  ?φ-aa(⌊*ArrVal*⌋)(⌊*?ε*(⌊*NTMap*⌋)(⌊*a*⌋)⌋) =
    (*?φ-aa* $\circ_{A\ cat\text{-}Set\ \alpha}$ *?φ-aa*$^{-1}{}_{C\ cat\text{-}Set\ \alpha}$)(⌊*ArrVal*⌋)(⌊$\mathfrak{C}$(⌊*CId*⌋)(⌊$\mathfrak{G}$(⌊*ObjMap*⌋)(⌊*a*⌋)⌋)⌋)
  by
    (
      cs-concl **cs-shallow**
        cs-simp:
          𝒵.cat-Set-Comp-ArrVal
          cat-Set-the-inverse[*symmetric*]
          cat-cs-simps adj-cs-simps cat-prod-cs-simps
        cs-intro:
          cat-arrow-cs-intros
          cat-cs-intros
          cat-op-intros
          adj-cs-intros
          cat-prod-cs-intros
)
also from
  is-cf-adjunction-axioms prems
  L.category-axioms R.category-axioms
  L.category-op R.category-op
  LR.is-functor-axioms RL.is-functor-axioms
  category-cat-Set
have ... = $\mathfrak{C}$(⌊*CId*⌋)(⌊$\mathfrak{G}$(⌊*ObjMap*⌋)(⌊*a*⌋)⌋)
  by
    (
      cs-concl
        cs-simp:
          cat-cs-simps
          cat-Set-components(*1*)
          category.cat-the-inverse-Comp-CId
        cs-intro:
          cat-arrow-cs-intros
          cat-cs-intros
          cat-op-intros
          cat-prod-cs-intros
)
finally have [*cat-cs-simps*]:
  ?φ-aa(⌊*ArrVal*⌋)(⌊*?ε*(⌊*NTMap*⌋)(⌊*a*⌋)⌋) = $\mathfrak{C}$(⌊*CId*⌋)(⌊$\mathfrak{G}$(⌊*ObjMap*⌋)(⌊*a*⌋)⌋)
  by *simp*
from
  prems is-cf-adjunction-axioms
  L.category-axioms R.category-axioms
show (($\mathfrak{G}$ $\circ_{CF-NTCF}$ *?ε*) $\cdot_{NTCF}$ (*?η* $\circ_{NTCF-CF}$ $\mathfrak{G}$))(⌊*NTMap*⌋)(⌊*a*⌋) = *ntcf-id* $\mathfrak{G}$(⌊*NTMap*⌋)(⌊*a*⌋)
  by
    (
      cs-concl
        cs-simp:
          cat-Set-the-inverse[*symmetric*]
          cf-adj-Comp-commute-RL
          cat-cs-simps

```
      adj-cs-simps
      cat-prod-cs-simps
      cat-op-simps
    cs-intro:
      cat-arrow-cs-intros
      cat-cs-intros
      adj-cs-intros
      cat-prod-cs-intros
      cat-op-intros
  )

 qed (auto intro: cat-cs-intros)

qed simp-all

lemmas [adj-cs-simps] = is-cf-adjunction.cf-adjunction-counit-unit

lemma (in is-cf-adjunction) cf-adjunction-unit-counit:
 (εC Φ ∘NTCF-CF 𝔉) ⋆NTCF (𝔉 ∘CF-NTCF ηC Φ) = ntcf-id 𝔉
 (is ‹(?ε ∘NTCF-CF 𝔉) ⋆NTCF (𝔉 ∘CF-NTCF ?η) = ntcf-id 𝔉›)
proof–
 from is-cf-adjunction-axioms have 𝔉η:
  𝔉 ∘CF-NTCF ?η : 𝔉 ↦CF 𝔉 ∘CF 𝔊 ∘CF 𝔉 : ℭ ↦↦Cα 𝔇
  by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)
 from is-cf-adjunction-axioms have ε𝔉:
  ?ε ∘NTCF-CF 𝔉 : 𝔉 ∘CF 𝔊 ∘CF 𝔉 ↦CF 𝔉 : ℭ ↦↦Cα 𝔇
  by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)
 from 𝔉η ε𝔉 have ε𝔉-𝔉η:
  (?ε ∘NTCF-CF 𝔉) ⋆NTCF (𝔉 ∘CF-NTCF ?η) : 𝔉 ↦CF 𝔉 : ℭ ↦↦Cα 𝔇
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
 from
  is-cf-adjunction.cf-adjunction-counit-unit[
    OF is-cf-adjunction-op,
    unfolded
      op-ntcf-cf-adjunction-unit[symmetric]
      op-ntcf-cf-adjunction-counit[symmetric]
      op-ntcf-cf-ntcf-comp[symmetric]
      op-ntcf-ntcf-cf-comp[symmetric]
      op-ntcf-ntcf-vcomp[symmetric]
      op-ntcf-ntcf-vcomp[symmetric, OF ε𝔉 𝔉η]
      LR.cf-ntcf-id-op-cf
    ]
 have
  op-ntcf (op-ntcf ((?ε ∘NTCF-CF 𝔉) ⋆NTCF (𝔉 ∘CF-NTCF ?η))) =
    op-ntcf (op-ntcf (ntcf-id 𝔉))
  by simp
 from this is-cf-adjunction-axioms ε𝔉-𝔉η show ?thesis
  by
    (
      cs-prems cs-shallow
        cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-prod-cs-intros
    )
qed

lemmas [adj-cs-simps] = is-cf-adjunction.cf-adjunction-unit-counit
```

13.7 Construction of an adjunction from universal morphisms from objects to functors

The subsection presents the construction of an adjunction given a structured collection of universal morphisms from objects to functors. The content of this subsection follows the statement and the proof of Theorem 2-i in Chapter IV-1 in [9].

13.7.1 The natural transformation associated with the adjunction constructed from universal morphisms from objects to functors

definition *cf-adjunction-AdjNT-of-unit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta$ =
    [
      $(\lambda cd\epsilon_{\circ}(op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!))\ \times_C\ \mathfrak{F}(\!|HomCod|\!))(\!|Obj|\!).$
        umap-of $\mathfrak{G}\ (cd(\!|0|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|cd(\!|0|\!)|\!))\ (\eta(\!|NTMap|\!)(\!|cd(\!|0|\!)|\!))\ (cd(\!|1_{\mathbb{N}}|\!))),$
      $Hom_{O.C\alpha}\mathfrak{F}(\!|HomCod|\!)(\mathfrak{F}-,-),$
      $Hom_{O.C\alpha}\mathfrak{F}(\!|HomDom|\!)(-,\mathfrak{G}-),$
      $op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!))\ \times_C\ (\mathfrak{F}(\!|HomCod|\!)),$
      cat-Set α
    ]$_{\circ}$

Components.

lemma *cf-adjunction-AdjNT-of-unit-components*:
  shows *cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTMap|\!)$ =
    (
      $\lambda cd\epsilon_{\circ}(op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!))\ \times_C\ \mathfrak{F}(\!|HomCod|\!))(\!|Obj|\!).$
        umap-of $\mathfrak{G}\ (cd(\!|0|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|cd(\!|0|\!)|\!))\ (\eta(\!|NTMap|\!)(\!|cd(\!|0|\!)|\!))\ \ (cd(\!|1_{\mathbb{N}}|\!))$
)
    and *cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTDom|\!) = Hom_{O.C\alpha}\mathfrak{F}(\!|HomCod|\!)(\mathfrak{F}-,-)$
    and *cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTCod|\!) = Hom_{O.C\alpha}\mathfrak{F}(\!|HomDom|\!)(-,\mathfrak{G}-)$
    and *cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTDGDom|\!)$ =
      $op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!))\ \times_C\ (\mathfrak{F}(\!|HomCod|\!))$
    and *cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTDGCod|\!) = cat\text{-}Set\ \alpha$
  unfolding *cf-adjunction-AdjNT-of-unit-def nt-field-simps*
  by (*simp-all add*: *nat-omega-simps*)

13.7.2 Natural transformation map

lemma *cf-adjunction-AdjNT-of-unit-NTMap-vsv*[*adj-cs-intros*]:
  vsv (*cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTMap|\!)$)
  unfolding *cf-adjunction-AdjNT-of-unit-components* **by** *simp*

lemma *cf-adjunction-AdjNT-of-unit-NTMap-vdomain*[*adj-cs-simps*]:
  assumes $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
  shows $\mathcal{D}_{\circ}$ (*cf-adjunction-AdjNT-of-unit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTMap|\!)$) = $(op\text{-}cat\ \mathfrak{C}\ \times_C\ \mathfrak{D})(\!|Obj|\!)$
proof–
  interpret *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
  show *?thesis*
    unfolding *cf-adjunction-AdjNT-of-unit-components*
    by (*simp add*: *cat-cs-simps*)
qed

lemma *cf-adjunction-AdjNT-of-unit-NTMap-app*[*adj-cs-simps*]:
  assumes $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $d \in_{\circ} \mathfrak{D}(\!|Obj|\!)$
  shows
    cf-adjunction-AdjNT-of-unit $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta(\!|NTMap|\!)(\!|c,\ d|\!)_{\bullet}$ =
      umap-of $\mathfrak{G}\ c\ (\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!))\ (\eta(\!|NTMap|\!)(\!|c|\!))\ d$

```
proof–
  interpret 𝔉: is-functor α ℭ 𝔇 𝔉 by (rule assms(1))
  from assms have [c, d]∘ ∈∘ (op-cat ℭ ×C 𝔇)(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  then show cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|) (|c, d|)• =
    umap-of 𝔊 c (𝔉(|ObjMap|)(|c|)) (η(|NTMap|)(|c|)) d
    unfolding cf-adjunction-AdjNT-of-unit-components
    by (simp add: nat-omega-simps cat-cs-simps)
qed

lemma cf-adjunction-AdjNT-of-unit-NTMap-vrange:
  assumes category α ℭ
    and category α 𝔇
    and 𝔉 : ℭ ↦↦CA 𝔇
    and 𝔊 : 𝔇 ↦↦CA ℭ
    and η : cf-id ℭ ↦CF 𝔊 ∘CF 𝔉 : ℭ ↦↦CA ℭ
  shows ℛ∘ (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|)) ⊆∘ cat-Set α(|Arr|)
proof–
  interpret 𝔉: is-functor α ℭ 𝔇 𝔉 by (rule assms(3))
  show ?thesis
  proof
    (
      rule vsv.vsv-vrange-vsubset,
      unfold cf-adjunction-AdjNT-of-unit-NTMap-vdomain[OF assms(3)]
    )
    show vsv (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|))
      by (intro adj-cs-intros)
    fix cd assume prems: cd ∈∘ (op-cat ℭ ×C 𝔇)(|Obj|)
    then obtain c d where cd-def: cd = [c, d]∘
      and c: c ∈∘ ℭ(|Obj|)
      and d: d ∈∘ 𝔇(|Obj|)
      by
        (
          auto
            simp: cat-op-simps
            elim:
              cat-prod-2-ObjE[OF 𝔉.HomDom.category-op 𝔉.HomCod.category-axioms]
        )
    from assms c d show
      cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|)(|cd|) ∈∘ cat-Set α(|Arr|)
      unfolding cd-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps adj-cs-simps cs-intro: cat-cs-intros
        )
  qed
qed
```

13.7.3 Adjunction constructed from universal morphisms from objects to functors is an adjunction

```
lemma cf-adjunction-AdjNT-of-unit-is-ntcf:
  assumes category α ℭ
```

```
    and category α 𝔇
    and 𝔉 : ℭ ↦↦_{Cα} 𝔇
    and 𝔊 : 𝔇 ↦↦_{Cα} ℭ
    and η : cf-id ℭ ↦_{CF} 𝔊 ∘_{CF} 𝔉 : ℭ ↦↦_{Cα} ℭ
  shows cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η :
    Hom_{O.Cα}𝔇(𝔉−,−) ↦_{CF} Hom_{O.Cα}ℭ(−,𝔊−) :
    op-cat ℭ ×_C 𝔇 ↦↦_{Cα} cat-Set α
proof–

  interpret ℭ: category α ℭ by (rule assms(1))
  interpret 𝔇: category α 𝔇 by (rule assms(2))
  interpret 𝔉: is-functor α ℭ 𝔇 𝔉 by (rule assms(3))
  interpret 𝔊: is-functor α 𝔇 ℭ 𝔊 by (rule assms(4))
  interpret η: is-ntcf α ℭ ℭ ‹cf-id ℭ› ‹𝔊 ∘_{CF} 𝔉› η by (rule assms(5))

  show ?thesis
  proof(intro is-ntcfI')

    show vfsequence (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η)
      unfolding cf-adjunction-AdjNT-of-unit-def by simp
    show vcard (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η) = 5_ℕ
      unfolding cf-adjunction-AdjNT-of-unit-def by (simp add: nat-omega-simps)
    from assms(2,3) show
      Hom_{O.Cα}𝔇(𝔉−,−) : op-cat ℭ ×_C 𝔇 ↦↦_{Cα} cat-Set α
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from assms show Hom_{O.Cα}ℭ(−,𝔊−) : op-cat ℭ ×_C 𝔇 ↦↦_{Cα} cat-Set α
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show vsv (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|))
      by (intro adj-cs-intros)
    from assms show
      𝒟_∘ (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|)) = (op-cat ℭ ×_C 𝔇)(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps adj-cs-simps)

    show cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|)(|cd|) :
      Hom_{O.Cα}𝔇(𝔉−,−)(|ObjMap|)(|cd|) ↦_{cat-Set α}
      Hom_{O.Cα}ℭ(−,𝔊−)(|ObjMap|)(|cd|)
      if cd ∈_∘ (op-cat ℭ ×_C 𝔇)(|Obj|) for cd
    proof–
      from that obtain c d
        where cd-def: cd = [c, d]_∘ and c: c ∈_∘ ℭ(|Obj|) and d: d ∈_∘ 𝔇(|Obj|)
        by
          (
            auto
              simp: cat-op-simps
              elim: cat-prod-2-ObjE[OF ℭ.category-op 𝔇.category-axioms]
          )
      from assms c d show ?thesis
        unfolding cd-def
        by
          (
            cs-concl cs-shallow
              cs-simp: adj-cs-simps cat-cs-simps cat-op-simps
              cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
          )
    qed

    show
      cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(|NTMap|)(|c′d′|) ∘_{A cat-Set α}
```

```
      Hom_{O.Cα}𝔇(𝔉−,−)(⌊ArrMap⌋)(⌊gf⌋) =
        Hom_{O.Cα}ℭ(−,𝔊−)(⌊ArrMap⌋)(⌊gf⌋) ∘_{A cat-Set α}
          cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η(⌊NTMap⌋)(⌊cd⌋)
    if gf : cd ↦_{op-cat ℭ ×_C 𝔇} c′d′ for cd c′d′ gf
  proof–
    from that obtain g f c c′ d d′
      where gf-def: gf = [g, f]∘
        and cd-def: cd = [c, d]∘
        and c′d′-def: c′d′ = [c′, d′]∘
        and g: g : c′ ↦_ℭ c
        and f: f : d ↦_𝔇 d′
      by
        (
          auto
            simp: cat-op-simps
            elim: cat-prod-2-is-arrE[OF ℭ.category-op 𝔇.category-axioms]
        )
    from assms g f that show ?thesis
      unfolding gf-def cd-def c′d′-def
      by
        (
          cs-concl
            cs-simp: cf-umap-of-cf-hom-unit-commute adj-cs-simps cat-cs-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
  qed

qed (auto simp: cf-adjunction-AdjNT-of-unit-components cat-cs-simps)

qed

lemma cf-adjunction-AdjNT-of-unit-is-ntcf'[adj-cs-intros]:
  assumes category α ℭ
    and category α 𝔇
    and 𝔉 : ℭ ↦↦_{Cα} 𝔇
    and 𝔊 : 𝔇 ↦↦_{Cα} ℭ
    and η : cf-id ℭ ↦_{CF} 𝔊 ∘_{CF} 𝔉 : ℭ ↦↦_{Cα} ℭ
    and 𝔖 = Hom_{O.Cα}𝔇(𝔉−,−)
    and 𝔖′ = Hom_{O.Cα}ℭ(−,𝔊−)
    and 𝔄 = op-cat ℭ ×_C 𝔇
    and 𝔅 = cat-Set α
  shows cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η : 𝔖 ↦_{CF} 𝔖′ : 𝔄 ↦↦_{Cα} 𝔅
  using assms(1–5) unfolding assms(6–9)
  by (rule cf-adjunction-AdjNT-of-unit-is-ntcf)
```

13.7.4 Adjunction constructed from universal morphisms from objects to functors

```
definition cf-adjunction-of-unit :: V ⇒ V ⇒ V ⇒ V ⇒ V
  where cf-adjunction-of-unit α 𝔉 𝔊 η =
    [𝔉, 𝔊, cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η]∘
```

Components.

```
lemma cf-adjunction-of-unit-components:
  shows [adj-cs-simps]: cf-adjunction-of-unit α 𝔉 𝔊 η(⌊AdjLeft⌋) = 𝔉
    and [adj-cs-simps]: cf-adjunction-of-unit α 𝔉 𝔊 η(⌊AdjRight⌋) = 𝔊
    and cf-adjunction-of-unit α 𝔉 𝔊 η(⌊AdjNT⌋) =
      cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η
  unfolding cf-adjunction-of-unit-def adj-field-simps
```

by (*simp-all add*: *nat-omega-simps*)

Natural transformation map.

lemma *cf-adjunction-of-unit-AdjNT-NTMap-vdomain*[*adj-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{D}_\circ$ (*cf-adjunction-of-unit* α $\mathfrak{F}$ $\mathfrak{G}$ $\eta(\!|AdjNT|\!)(\!|NTMap|\!)$) =
($op\text{-}cat$ $\mathfrak{C}$ $\times_C$ $\mathfrak{D}$)$(\!|Obj|\!)$
using *assms*
unfolding *cf-adjunction-of-unit-components*(*3*)
by (*rule cf-adjunction-AdjNT-of-unit-NTMap-vdomain*)

lemma *cf-adjunction-of-unit-AdjNT-NTMap-app*[*adj-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $d \in_\circ \mathfrak{D}(\!|Obj|\!)$
shows
cf-adjunction-of-unit α $\mathfrak{F}$ $\mathfrak{G}$ $\eta(\!|AdjNT|\!)(\!|NTMap|\!)(\!|c,\ d|\!)_\bullet$ =
umap-of $\mathfrak{G}$ c ($\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)$) ($\eta(\!|NTMap|\!)(\!|c|\!)$) d
using *assms*
unfolding *cf-adjunction-of-unit-components*(*3*)
by (*rule cf-adjunction-AdjNT-of-unit-NTMap-app*)

The adjunction constructed from universal morphisms from objects to functors is an adjunction.

lemma *cf-adjunction-of-unit-is-cf-adjunction*:
assumes *category* α $\mathfrak{C}$
and *category* α $\mathfrak{D}$
and $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and η : *cf-id* $\mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge x.\ x \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow$ *universal-arrow-of* $\mathfrak{G}$ x ($\mathfrak{F}(\!|ObjMap|\!)(\!|x|\!)$) ($\eta(\!|NTMap|\!)(\!|x|\!)$)
shows *cf-adjunction-of-unit* α $\mathfrak{F}$ $\mathfrak{G}$ $\eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
and η_C (*cf-adjunction-of-unit* α $\mathfrak{F}$ $\mathfrak{G}$ η) = η
proof–

interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{D}$: *category* α $\mathfrak{D}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{D}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*4*))
interpret η: *is-ntcf* α $\mathfrak{C}$ $\mathfrak{C}$ ‹*cf-id* $\mathfrak{C}$› ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› η **by** (*rule assms*(*5*))

show *caou-η*: *cf-adjunction-of-unit* α $\mathfrak{F}$ $\mathfrak{G}$ $\eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
proof
(
intro
is-cf-adjunctionI[*OF - - assms*(*1–4*)]
is-iso-ntcf-if-bnt-proj-snd-is-iso-ntcf[
OF $\mathfrak{C}$.*category-op* $\mathfrak{D}$.*category-axioms*
],
unfold cat-op-simps cf-adjunction-of-unit-components
)
show *caou-η*: *cf-adjunction-AdjNT-of-unit* α $\mathfrak{F}$ $\mathfrak{G}$ η :
$Hom_{O.C\alpha}\mathfrak{D}(\mathfrak{F}-,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$:
$op\text{-}cat$ $\mathfrak{C}$ $\times_C$ $\mathfrak{D}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
unfolding *cf-adjunction-of-unit-components*
by (*rule cf-adjunction-AdjNT-of-unit-is-ntcf*[*OF assms*(*1–5*)])
fix a **assume** *prems*: $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
have *ua-of-ηa*:
ntcf-ua-of α $\mathfrak{G}$ a ($\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$) ($\eta(\!|NTMap|\!)(\!|a|\!)$) :
$Hom_{O.C\alpha}\mathfrak{D}(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!),-) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{C}(a,-) \circ_{CF} \mathfrak{G}$:
$\mathfrak{D} \mapsto\mapsto_{C\alpha}$ *cat-Set* α

```
    by
      (
        rule is-functor.cf-ntcf-ua-of-is-iso-ntcf[
          OF assms(4) assms(6)[OF prems]
          ]
      )
  have [adj-cs-simps]:
    cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η_op-cat 𝔈,𝔇(a,−)_NTCF =
      ntcf-ua-of α 𝔊 a (𝔉⦇ObjMap⦈⦇a⦈) (η⦇NTMap⦈⦇a⦈)
  proof(rule ntcf-eqI)
    from assms(1–5) caou-η prems show lhs:
      cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η_op-cat 𝔈,𝔇(a,−)_NTCF :
        Hom_O.Cα𝔇(𝔉⦇ObjMap⦈⦇a⦈,−) ↦_CF Hom_O.Cα𝔈(a,−) ∘_CF 𝔊 :
        𝔇 ↦↦_Cα cat-Set α
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
        )
      from ua-of-ηa show rhs:
        ntcf-ua-of α 𝔊 a (𝔉⦇ObjMap⦈⦇a⦈) (η⦇NTMap⦈⦇a⦈) :
          Hom_O.Cα𝔇(𝔉⦇ObjMap⦈⦇a⦈,−) ↦_CF Hom_O.Cα𝔈(a,−) ∘_CF 𝔊 :
          𝔇 ↦↦_Cα cat-Set α
        by (cs-concl cs-shallow cs-intro: ntcf-cs-intros)
      from lhs have dom-lhs:
        𝒟∘ ((cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η_op-cat 𝔈,𝔇(a,−)_NTCF)⦇NTMap⦈) =
          𝔇⦇Obj⦈
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      from lhs assms(4) have dom-rhs:
        𝒟∘ (ntcf-ua-of α 𝔊 a (𝔉⦇ObjMap⦈⦇a⦈) (η⦇NTMap⦈⦇a⦈)⦇NTMap⦈) = 𝔇⦇Obj⦈
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      show
        (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η_op-cat 𝔈,𝔇(a,−)_NTCF)⦇NTMap⦈ =
          ntcf-ua-of α 𝔊 a (𝔉⦇ObjMap⦈⦇a⦈) (η⦇NTMap⦈⦇a⦈)⦇NTMap⦈
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
        fix d assume prems′: d ∈∘ 𝔇⦇Obj⦈
        from assms(3,4) prems prems′ show
          (cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η_op-cat 𝔈,𝔇(a,−)_NTCF)⦇NTMap⦈⦇d⦈ =
            ntcf-ua-of α 𝔊 a (𝔉⦇ObjMap⦈⦇a⦈) (η⦇NTMap⦈⦇a⦈)⦇NTMap⦈⦇d⦈
          by (cs-concl cs-shallow cs-simp: adj-cs-simps cat-cs-simps)
      qed (simp-all add: bnt-proj-snd-NTMap-vsv 𝔊.ntcf-ua-of-NTMap-vsv)
    qed simp-all
    from assms(1–5) assms(6)[OF prems] prems show
      cf-adjunction-AdjNT-of-unit α 𝔉 𝔊 η_op-cat 𝔈,𝔇(a,−)_NTCF :
        Hom_O.Cα𝔇(𝔉−,−)_op-cat 𝔈,𝔇(a,−)_CF ↦_CF.iso
        Hom_O.Cα𝔈(−,𝔊−)_op-cat 𝔈,𝔇(a,−)_CF :
        𝔇 ↦↦_Cα cat-Set α
      by
        (
          cs-concl cs-shallow
            cs-simp: adj-cs-simps cat-cs-simps cs-intro: cat-cs-intros
        )
  qed (auto simp: cf-adjunction-of-unit-def nat-omega-simps)

  show η_C (cf-adjunction-of-unit α 𝔉 𝔊 η) = η
  proof(rule ntcf-eqI)
```

```
    from caou-η show lhs:
      η_C (cf-adjunction-of-unit α 𝔉 𝔊 η) :
        cf-id ℭ ↦_CF 𝔊 ∘_CF 𝔉 : ℭ ↦↦_Cα ℭ
      by (cs-concl cs-shallow cs-intro: adj-cs-intros)
    show rhs: η : cf-id ℭ ↦_CF 𝔊 ∘_CF 𝔉 : ℭ ↦↦_Cα ℭ
      by (auto intro: cat-cs-intros)
    from lhs have dom-lhs:
      𝒟∘ (η_C (cf-adjunction-of-unit α 𝔉 𝔊 η)(⌊NTMap⌋)) = ℭ(⌊Obj⌋)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    have dom-rhs: 𝒟∘ (η(⌊NTMap⌋)) = ℭ(⌊Obj⌋) by (auto simp: cat-cs-simps)
    show η_C (cf-adjunction-of-unit α 𝔉 𝔊 η)(⌊NTMap⌋) = η(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume prems: a ∈∘ ℭ(⌊Obj⌋)
      from assms(1–5) prems caou-η show
        η_C (cf-adjunction-of-unit α 𝔉 𝔊 η)(⌊NTMap⌋)(⌊a⌋) = η(⌊NTMap⌋)(⌊a⌋)
        by
          (
            cs-concl cs-shallow
              cs-simp:
                adj-cs-simps cat-cs-simps cf-adjunction-of-unit-components(3)
              cs-intro: cat-cs-intros
          )
    qed (auto intro: adj-cs-intros)
  qed simp-all

qed
```

13.8 Construction of an adjunction from a functor and universal morphisms from objects to functors

The subsection presents the construction of an adjunction given a functor and a structured collection of universal morphisms from objects to functors. The content of this subsection follows the statement and the proof of Theorem 2-ii in Chapter IV-1 in [9].

13.8.1 Left adjoint

```
definition cf-la-of-ra :: (V ⇒ V) ⇒ V ⇒ V ⇒ V
  where cf-la-of-ra F 𝔊 η =
    [
      (λx∈∘𝔊(⌊HomCod⌋)(⌊Obj⌋). F x),
      (
        λh∈∘𝔊(⌊HomCod⌋)(⌊Arr⌋). THE f'.
          f' : F (𝔊(⌊HomCod⌋)(⌊Dom⌋)(⌊h⌋)) ↦_𝔊(⌊HomDom⌋) F (𝔊(⌊HomCod⌋)(⌊Cod⌋)(⌊h⌋)) ∧
            η(⌊𝔊(⌊HomCod⌋)(⌊Cod⌋)(⌊h⌋)⌋) ∘_A𝔊(⌊HomCod⌋) h =
              (
                umap-of
                  𝔊
                  (𝔊(⌊HomCod⌋)(⌊Dom⌋)(⌊h⌋))
                  (F (𝔊(⌊HomCod⌋)(⌊Dom⌋)(⌊h⌋)))
                  (η(⌊𝔊(⌊HomCod⌋)(⌊Dom⌋)(⌊h⌋)⌋))
                  (F (𝔊(⌊HomCod⌋)(⌊Cod⌋)(⌊h⌋)))
              )(⌊ArrVal⌋)(⌊f'⌋)
      ),
      𝔊(⌊HomCod⌋),
      𝔊(⌊HomDom⌋)
    ]∘
```

Components.

lemma *cf-la-of-ra-components*:
shows *cf-la-of-ra F 𝔊 η(|ObjMap|) = (λx∈∘𝔊(|HomCod|)(|Obj|). F x)*
and *cf-la-of-ra F 𝔊 η(|ArrMap|) =*
(
λh∈∘𝔊(|HomCod|)(|Arr|). THE f′.
f′ : F (𝔊(|HomCod|)(|Dom|)(|h|)) ↦_{𝔊(|HomDom|)} F (𝔊(|HomCod|)(|Cod|)(|h|)) ∧
η(|𝔊(|HomCod|)(|Cod|)(|h|)|) ∘_{A𝔊(|HomCod|)} h =
(
umap-of
𝔊
(𝔊(|HomCod|)(|Dom|)(|h|))
(F (𝔊(|HomCod|)(|Dom|)(|h|)))
(η(|𝔊(|HomCod|)(|Dom|)(|h|)|))
(F (𝔊(|HomCod|)(|Cod|)(|h|)))
)(|ArrVal|)(|f′|)
)
and *cf-la-of-ra F 𝔊 η(|HomDom|) = 𝔊(|HomCod|)*
and *cf-la-of-ra F 𝔊 η(|HomCod|) = 𝔊(|HomDom|)*
unfolding *cf-la-of-ra-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

13.8.2 Object map

mk-VLambda *cf-la-of-ra-components*(*1*)
|*vsv cf-la-of-ra-ObjMap-vsv*[*adj-cs-intros*]|

mk-VLambda (**in** *is-functor*)
cf-la-of-ra-components(*1*)[**where** *?𝔊=𝔉*, *unfolded cf-HomCod*]
|*vdomain cf-la-of-ra-ObjMap-vdomain*[*adj-cs-simps*]|
|*app cf-la-of-ra-ObjMap-app*[*adj-cs-simps*]|

lemmas [*adj-cs-simps*] =
is-functor.cf-la-of-ra-ObjMap-vdomain
is-functor.cf-la-of-ra-ObjMap-app

13.8.3 Arrow map

mk-VLambda *cf-la-of-ra-components*(*2*)
|*vsv cf-la-of-ra-ArrMap-vsv*[*adj-cs-intros*]|

mk-VLambda (**in** *is-functor*)
cf-la-of-ra-components(*2*)[**where** *?𝔊=𝔉*, *unfolded cf-HomCod cf-HomDom*]
|*vdomain cf-la-of-ra-ArrMap-vdomain*[*adj-cs-simps*]|
|*app cf-la-of-ra-ArrMap-app*|

lemmas [*adj-cs-simps*] = *is-functor.cf-la-of-ra-ArrMap-vdomain*

lemma (**in** *is-functor*) *cf-la-of-ra-ArrMap-app′*:
assumes *h : a ↦_𝔄 b*
shows
cf-la-of-ra F 𝔉 η(|ArrMap|)(|h|) =
(
THE f′.
f′ : F a ↦_𝔅 F b ∧
η(|b|) ∘_{A𝔅} h = umap-of 𝔉 a (F a) (η(|a|)) (F b)(|ArrVal|)(|f′|)
)
proof–
from *assms* **have** *h*: *h ∈∘ 𝔄(|Arr|)* **by** (*simp add*: *cat-cs-intros*)

```
  from assms have h-Dom: 𝔅(⦇Dom⦈)(⦇h⦈) = a and h-Cod: 𝔅(⦇Cod⦈)(⦇h⦈) = b
    by (simp-all add: cat-cs-simps)
  show ?thesis by (rule cf-la-of-ra-ArrMap-app[OF h, unfolded h-Dom h-Cod])
qed

lemma cf-la-of-ra-ArrMap-app-unique:
  assumes 𝔊 : 𝔇 ↦↦_{Cα} ℭ
    and f : a ↦_ℭ b
    and universal-arrow-of 𝔊 a (cf-la-of-ra F 𝔊 η(⦇ObjMap⦈)(⦇a⦈)) (η(⦇a⦈))
    and universal-arrow-of 𝔊 b (cf-la-of-ra F 𝔊 η(⦇ObjMap⦈)(⦇b⦈)) (η(⦇b⦈))
  shows cf-la-of-ra F 𝔊 η(⦇ArrMap⦈)(⦇f⦈) : F a ↦_𝔇 F b
    and η(⦇b⦈) ∘_{Aℭ} f =
      umap-of 𝔊 a (F a) (η(⦇a⦈)) (F b)(⦇ArrVal⦈)(⦇cf-la-of-ra F 𝔊 η(⦇ArrMap⦈)(⦇f⦈)⦈)
    and ⋀f'.
      ⟦
        f' : F a ↦_𝔇 F b;
        η(⦇b⦈) ∘_{Aℭ} f = umap-of 𝔊 a (F a) (η(⦇a⦈)) (F b)(⦇ArrVal⦈)(⦇f'⦈)
      ⟧ ⟹ cf-la-of-ra F 𝔊 η(⦇ArrMap⦈)(⦇f⦈) = f'
proof–

  interpret 𝔊: is-functor α 𝔇 ℭ 𝔊 by (rule assms(1))

  from assms(2) have a: a ∈₀ ℭ(⦇Obj⦈) and b: b ∈₀ ℭ(⦇Obj⦈)
    by (simp-all add: cat-cs-intros)
  note ua-η-a = 𝔊.universal-arrow-ofD[OF assms(3)]
  note ua-η-b = 𝔊.universal-arrow-ofD[OF assms(4)]
  from ua-η-b(2) have [cat-cs-intros]:
    ⟦ c = b; c' = 𝔊(⦇ObjMap⦈)(⦇cf-la-of-ra F 𝔊 η(⦇ObjMap⦈)(⦇b⦈)⦈) ⟧ ⟹ η(⦇b⦈) : c ↦_ℭ c'
    for c c'
    by auto
  from assms(1,2) ua-η-a(2) have ηa-f:
    η(⦇b⦈) ∘_{Aℭ} f : a ↦_ℭ 𝔊(⦇ObjMap⦈)(⦇cf-la-of-ra F 𝔊 η(⦇ObjMap⦈)(⦇b⦈)⦈)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms(1,2) have lara-a: cf-la-of-ra F 𝔊 η(⦇ObjMap⦈)(⦇a⦈) = F a
    and lara-b: cf-la-of-ra F 𝔊 η(⦇ObjMap⦈)(⦇b⦈) = F b
    by (cs-concl cs-simp: adj-cs-simps cs-intro: cat-cs-intros)+

  from theD
    [
      OF
        ua-η-a(3)[OF ua-η-b(1) ηa-f, unfolded lara-a lara-b]
        𝔊.cf-la-of-ra-ArrMap-app'[OF assms(2), of F η]
    ]
  show cf-la-of-ra F 𝔊 η(⦇ArrMap⦈)(⦇f⦈) : F a ↦_𝔇 F b
    and η(⦇b⦈) ∘_{Aℭ} f = umap-of
      𝔊 a (F a) (η(⦇a⦈)) (F b)(⦇ArrVal⦈)(⦇cf-la-of-ra F 𝔊 η(⦇ArrMap⦈)(⦇f⦈)⦈)
    and ⋀f'.
      ⟦
        f' : F a ↦_𝔇 F b;
        η(⦇b⦈) ∘_{Aℭ} f = umap-of 𝔊 a (F a) (η(⦇a⦈)) (F b)(⦇ArrVal⦈)(⦇f'⦈)
      ⟧ ⟹ cf-la-of-ra F 𝔊 η(⦇ArrMap⦈)(⦇f⦈) = f'
    by blast+

qed

lemma cf-la-of-ra-ArrMap-app-is-arr[adj-cs-intros]:
  assumes 𝔊 : 𝔇 ↦↦_{Cα} ℭ
    and f : a ↦_ℭ b
```

and *universal-arrow-of* $\mathfrak{G}$ a (*cf-la-of-ra* F $\mathfrak{G}$ $\eta(\!|ObjMap|\!)(\!|a|\!)$) ($\eta(\!|a|\!)$)
and *universal-arrow-of* $\mathfrak{G}$ b (*cf-la-of-ra* F $\mathfrak{G}$ $\eta(\!|ObjMap|\!)(\!|b|\!)$) ($\eta(\!|b|\!)$)
and $Fa = F\ a$
and $Fb = F\ b$
shows *cf-la-of-ra* F $\mathfrak{G}$ $\eta(\!|ArrMap|\!)(\!|f|\!) : Fa \mapsto_{\mathfrak{D}} Fb$
using *assms*(*1–4*) **unfolding** *assms*(*5,6*) **by** (*rule cf-la-of-ra-ArrMap-app-unique*)

13.8.4 An adjunction constructed from a functor and universal morphisms from objects to functors is an adjunction

lemma *cf-la-of-ra-is-functor*:
assumes $\mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge c.\ c \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow F\ c \in_{\circ} \mathfrak{D}(\!|Obj|\!)$
and $\bigwedge c.\ c \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow$
universal-arrow-of $\mathfrak{G}$ c (*cf-la-of-ra* F $\mathfrak{G}$ $\eta(\!|ObjMap|\!)(\!|c|\!)$) ($\eta(\!|c|\!)$)
and $\bigwedge c\ c'\ h.\ h : c \mapsto_{\mathfrak{C}} c' \Longrightarrow$
$\mathfrak{G}(\!|ArrMap|\!)(\!|$*cf-la-of-ra* F $\mathfrak{G}$ $\eta(\!|ArrMap|\!)(\!|h|\!)|\!) \circ_{A\mathfrak{C}} \eta(\!|c|\!) = \eta(\!|c'|\!) \circ_{A\mathfrak{C}} h$
shows *cf-la-of-ra* F $\mathfrak{G}$ $\eta : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ (**is** $\langle ?\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}\rangle$)
proof–

interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{D}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))

show *cf-la-of-ra* F $\mathfrak{G}$ $\eta : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof(*rule is-functorI'*)

show *vfsequence* $?\mathfrak{F}$ **unfolding** *cf-la-of-ra-def* **by** *auto*
show *vcard* $?\mathfrak{F} = \mathcal{4}_{\mathbb{N}}$
unfolding *cf-la-of-ra-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ ($?\mathfrak{F}(\!|ObjMap|\!)$) $\subseteq_{\circ} \mathfrak{D}(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold* $\mathfrak{G}$.*cf-la-of-ra-ObjMap-vdomain*)
fix x **assume** $x \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
with *assms*(*1*) **show** $?\mathfrak{F}(\!|ObjMap|\!)(\!|x|\!) \in_{\circ} \mathfrak{D}(\!|Obj|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *adj-cs-simps* **cs-intro**: *assms*(*2*))
qed (*auto intro*: *adj-cs-intros*)

show $?\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) : ?\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{D}} ?\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
if $f : a \mapsto_{\mathfrak{C}} b$ **for** $a\ b\ f$
proof–
from *that* **have** *a*: $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** *b*: $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
by (*simp-all add*: *cat-cs-intros*)
have *ua-η-a*: *universal-arrow-of* $\mathfrak{G}$ a ($?\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$) ($\eta(\!|a|\!)$)
and *ua-η-b*: *universal-arrow-of* $\mathfrak{G}$ b ($?\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$) ($\eta(\!|b|\!)$)
by (*intro assms*(*3*)[*OF a*] *assms*(*3*)[*OF b*])+
from *a b cf-la-of-ra-ArrMap-app-unique*(*1*)[*OF assms*(*1*) *that ua-η-a ua-η-b*]
show *?thesis*
by (*cs-concl* **cs-shallow cs-simp**: *adj-cs-simps*)
qed

show $?\mathfrak{F}(\!|ArrMap|\!)(\!|g \circ_{A\mathfrak{C}} f|\!) = ?\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{D}} ?\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
if $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **for** $b\ c\ g\ a\ f$
proof–

from *that* **have** *a*: $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** *b*: $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** *c*: $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
by (*simp-all add*: *cat-cs-intros*)
from *assms*(*1*) *that* **have** *gf*: $g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

note *ua-η-a* = *assms*(*3*)[*OF a*]

```
  and ua-η-b = assms(3)[OF b]
  and ua-η-c = assms(3)[OF c]

 note lara-f =
  cf-la-of-ra-ArrMap-app-unique[OF assms(1) that(2) ua-η-a ua-η-b]
 note lara-g =
  cf-la-of-ra-ArrMap-app-unique[OF assms(1) that(1) ua-η-b ua-η-c]
 note lara-gf =
  cf-la-of-ra-ArrMap-app-unique[OF assms(1) gf ua-η-a ua-η-c]

 note ua-η-a = 𝔊.universal-arrow-ofD[OF ua-η-a]
  and ua-η-b = 𝔊.universal-arrow-ofD[OF ua-η-b]
  and ua-η-c = 𝔊.universal-arrow-ofD[OF ua-η-c]

 from ua-η-a(2) assms(1) that have ηa:
  η(|a|) : a ↦_ℭ 𝔊(|ObjMap|)(|F a|)
  by (cs-prems cs-simp: adj-cs-simps cs-intro: cat-cs-intros)
 from ua-η-b(2) assms(1) that have ηb:
  η(|b|) : b ↦_ℭ 𝔊(|ObjMap|)(|F b|)
  by (cs-prems cs-shallow cs-simp: adj-cs-simps cs-intro: cat-cs-intros)
 from ua-η-c(2) assms(1) that have ηc:
  η(|c|) : c ↦_ℭ 𝔊(|ObjMap|)(|F c|)
  by (cs-prems cs-shallow cs-simp: adj-cs-simps cs-intro: cat-cs-intros)

 from assms(1) that ηc have
  η(|c|) ∘_Aℭ (g ∘_Aℭ f) = (η(|c|) ∘_Aℭ g) ∘_Aℭ f
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 also from assms(1) lara-g(1) that(2) ηb have
  ... = 𝔊(|ArrMap|)(|?𝔉(|ArrMap|)(|g|)|) ∘_Aℭ (η(|b|) ∘_Aℭ f)
  by
    (
      cs-concl
        cs-simp: lara-g(2) cat-cs-simps
        cs-intro: cat-cs-intros cat-prod-cs-intros
    )
 also from assms(1) lara-f(1) ηa have ... =
  𝔊(|ArrMap|)(|?𝔉(|ArrMap|)(|g|)|) ∘_Aℭ
    (𝔊(|ArrMap|)(|?𝔉(|ArrMap|)(|f|)|) ∘_Aℭ η(|a|))
  by (cs-concl cs-shallow cs-simp: lara-f(2) cat-cs-simps)
 finally have [symmetric, cat-cs-simps]:
  η(|c|) ∘_Aℭ (g ∘_Aℭ f) = ....
 from assms(1) this ηa ηb ηc lara-g(1) lara-f(1) have
  η(|c|) ∘_Aℭ (g ∘_Aℭ f) =
    umap-of 𝔊 a (F a) (η(|a|)) (F c)(|ArrVal|)(|?𝔉(|ArrMap|)(|g|) ∘_A𝔇
    ?𝔉(|ArrMap|)(|f|)|)
  by
    (
      cs-concl cs-shallow
        cs-simp: adj-cs-simps cat-cs-simps
        cs-intro: adj-cs-intros cat-cs-intros
    )
 moreover from assms(1) lara-g(1) lara-f(1) have
  ?𝔉(|ArrMap|)(|g|) ∘_A𝔇 ?𝔉(|ArrMap|)(|f|) : F a ↦_𝔇 F c
  by (cs-concl cs-shallow cs-intro: adj-cs-intros cat-cs-intros)
 ultimately show ?thesis by (intro lara-gf(3))

qed
```

```
    show ?𝔉(|ArrMap|)(|ℭ(|CId|)(|c|)|) = 𝔇(|CId|)(|?𝔉(|ObjMap|)(|c|)|) if c ∈∘ ℭ(|Obj|) for c
    proof–
      note lara-c = cf-la-of-ra-ArrMap-app-unique[
          OF
            assms(1)
            𝔊.HomCod.cat-CId-is-arr[OF that]
            assms(3)[OF that]
            assms(3)[OF that]
          ]
      from assms(1) that have 𝔇c: 𝔇(|CId|)(|F c|) : F c ↦_𝔇 F c
        by (cs-concl cs-simp: cat-cs-simps cs-intro: assms(2) cat-cs-intros)
      from 𝔊.universal-arrow-ofD(2)[OF assms(3)[OF that]] assms(1) that have ηc:
        η(|c|) : c ↦_ℭ 𝔊(|ObjMap|)(|F c|)
        by (cs-prems cs-shallow cs-simp: adj-cs-simps cs-intro: cat-cs-intros)
      from assms(1) that ηc have
        η(|c|) ∘_Aℭ ℭ(|CId|)(|c|) =
          umap-of 𝔊 c (F c) (η(|c|)) (F c)(|ArrVal|)(|𝔇(|CId|)(|F c|)|)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: assms(2) cat-cs-intros)
      note [cat-cs-simps] = lara-c(3)[OF 𝔇c this]
      from assms(1) that 𝔇c show ?thesis
        by
          (
          cs-concl cs-shallow
            cs-simp: adj-cs-simps cat-cs-simps cs-intro: cat-cs-intros
          )
    qed
  qed (auto simp: cf-la-of-ra-components cat-cs-intros cat-cs-simps)

qed

lemma cf-la-of-ra-is-ntcf:
  fixes F ℭ 𝔉 𝔊 η_m η
  defines 𝔉 ≡ cf-la-of-ra F 𝔊 η_m
    and η ≡ [η_m, cf-id ℭ, 𝔊 ∘_CF 𝔉, ℭ, ℭ]∘
  assumes 𝔊 : 𝔇 ↦↦_Cα ℭ
    and ⋀c. c ∈∘ ℭ(|Obj|) ⟹ F c ∈∘ 𝔇(|Obj|)
    and ⋀c. c ∈∘ ℭ(|Obj|) ⟹ universal-arrow-of 𝔊 c (𝔉(|ObjMap|)(|c|)) (η(|NTMap|)(|c|))
    and ⋀c c′ h. h : c ↦_ℭ c′ ⟹
      𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|h|)|) ∘_Aℭ (η(|NTMap|)(|c|)) = (η(|NTMap|)(|c′|)) ∘_Aℭ h
    and vsv (η(|NTMap|))
    and 𝒟∘ (η(|NTMap|)) = ℭ(|Obj|)
  shows η : cf-id ℭ ↦_CF 𝔊 ∘_CF 𝔉 : ℭ ↦↦_Cα ℭ
proof–
  interpret 𝔊: is-functor α 𝔇 ℭ 𝔊 by (rule assms(3))
  have η-components:
    η(|NTMap|) = η_m
    η(|NTDom|) = cf-id ℭ
    η(|NTCod|) = 𝔊 ∘_CF 𝔉
    η(|NTDGDom|) = ℭ
    η(|NTDGCod|) = ℭ
    unfolding η-def nt-field-simps by (simp-all add: nat-omega-simps)
  note 𝔉-def = 𝔉-def[folded η-components(1)]
  have 𝔉: 𝔉 : ℭ ↦↦_Cα 𝔇
    unfolding 𝔉-def
    by (auto intro: cf-la-of-ra-is-functor[OF assms(3–6)[unfolded 𝔉-def]])
  show η : cf-id ℭ ↦_CF 𝔊 ∘_CF 𝔉 : ℭ ↦↦_Cα ℭ
  proof(rule is-ntcfI′)
    show vfsequence η unfolding η-def by simp
```

```
    show vcard η = 5ₙ unfolding η-def by (simp-all add: nat-omega-simps)
    from assms(2) show cf-id 𝔈 : 𝔈 ↦↦_Cα 𝔈
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(2) 𝔉 show 𝔊 ∘_CF 𝔉 : 𝔈 ↦↦_Cα 𝔈
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show η(|NTMap|)(|a|) : cf-id 𝔈(|ObjMap|)(|a|) ↦_𝔈 (𝔊 ∘_CF 𝔉)(|ObjMap|)(|a|)
      if a ∈₀ 𝔈(|Obj|) for a
      using assms(2) 𝔉 that 𝔊.universal-arrow-ofD(2)[OF assms(5)[OF that]]
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show
      η(|NTMap|)(|b|) ∘_A𝔈 cf-id 𝔈(|ArrMap|)(|f|) =
        (𝔊 ∘_CF 𝔉)(|ArrMap|)(|f|) ∘_A𝔈 η(|NTMap|)(|a|)
      if f : a ↦_𝔈 b for a b f
      using assms(3) 𝔉 that
      by
        (
          cs-concl cs-shallow
            cs-simp: assms(6) cat-cs-simps cs-intro: cat-cs-intros
        )
  qed (auto simp: η-components(2–5) assms(7–8))
qed

lemma cf-la-of-ra-is-unit:
  fixes F 𝔈 𝔉 𝔊 η_m η
  defines 𝔉 ≡ cf-la-of-ra F 𝔊 η_m
    and η ≡ [η_m, cf-id 𝔈, 𝔊 ∘_CF 𝔉, 𝔈, 𝔈]₀
  assumes category α 𝔈
    and category α 𝔇
    and 𝔊 : 𝔇 ↦↦_Cα 𝔈
    and ⋀c. c ∈₀ 𝔈(|Obj|) ⟹ F c ∈₀ 𝔇(|Obj|)
    and ⋀c. c ∈₀ 𝔈(|Obj|) ⟹
      universal-arrow-of 𝔊 c (𝔉(|ObjMap|)(|c|)) (η(|NTMap|)(|c|))
    and ⋀c c′ h. h : c ↦_𝔈 c′ ⟹
      𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|h|)|) ∘_A𝔈 (η(|NTMap|)(|c|)) = (η(|NTMap|)(|c′|)) ∘_A𝔈 h
    and vsv (η(|NTMap|))
    and 𝒟₀ (η(|NTMap|)) = 𝔈(|Obj|)
  shows cf-adjunction-of-unit α 𝔉 𝔊 η : 𝔉 ⇌_CF 𝔊 : 𝔈 ⇌⇌_Cα 𝔇
    and η_C (cf-adjunction-of-unit α 𝔉 𝔊 η) = η
proof–
  have η-components:
    η(|NTMap|) = η_m
    η(|NTDom|) = cf-id 𝔈
    η(|NTCod|) = 𝔊 ∘_CF 𝔉
    η(|NTDGDom|) = 𝔈
    η(|NTDGCod|) = 𝔈
    unfolding η-def nt-field-simps by (simp-all add: nat-omega-simps)
  note 𝔉-def = 𝔉-def[folded η-components(1)]
  note 𝔉 = cf-la-of-ra-is-functor
    [
      where F=F and 𝔈=𝔈 and 𝔊=𝔊 and η=η_m,
      folded 𝔉-def[unfolded η-components(1)],
      folded η-components(1),
      OF assms(5–8),
      simplified
    ]
  note η = cf-la-of-ra-is-ntcf
    [
      where F=F and 𝔈=𝔈 and 𝔊=𝔊 and η_m=η_m,
```

```
    folded 𝔉-def[unfolded η-components(1)],
    folded η-def,
    OF assms(5–10),
    simplified
  ]
 show cf-adjunction-of-unit α 𝔉 𝔊 η : 𝔉 ⇌_CF 𝔊 : ℭ ⇌⇌_Cα 𝔇
  and η_C (cf-adjunction-of-unit α 𝔉 𝔊 η) = η
  by
   (
    intro cf-adjunction-of-unit-is-cf-adjunction[
     OF assms(3,4) 𝔉 assms(5) η assms(7), simplified, folded 𝔉-def
     ]
   )+
qed
```

13.9 Construction of an adjunction from universal morphisms from functors to objects

13.9.1 Definition and elementary properties

The subsection presents the construction of an adjunction given a structured collection of universal morphisms from functors to objects. The content of this subsection follows the statement and the proof of Theorem 2-iii in Chapter IV-1 in [9].

```
definition cf-adjunction-of-counit :: V ⇒ V ⇒ V ⇒ V ⇒ V
 where cf-adjunction-of-counit α 𝔉 𝔊 ε =
  op-cf-adj (cf-adjunction-of-unit α (op-cf 𝔊) (op-cf 𝔉) (op-ntcf ε))
```

Components.

```
lemma cf-adjunction-of-counit-components:
 shows cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjLeft|) = op-cf (op-cf 𝔉)
  and cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjRight|) = op-cf (op-cf 𝔊)
  and cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjNT|) = op-cf-adj-nt
   (op-cf 𝔊(|HomDom|))
   (op-cf 𝔊(|HomCod|))
   (cf-adjunction-AdjNT-of-unit α (op-cf 𝔊) (op-cf 𝔉) (op-ntcf ε))
 unfolding
  cf-adjunction-of-counit-def
  op-cf-adj-components
  cf-adjunction-of-unit-components
 by (simp-all add: cat-op-simps)
```

13.9.2 Natural transformation map

```
lemma cf-adjunction-of-counit-NTMap-vsv:
 vsv (cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjNT|)(|NTMap|))
 unfolding cf-adjunction-of-counit-components by (rule inv-ntcf-NTMap-vsv)
```

13.9.3 An adjunction constructed from universal morphisms from functors to objects is an adjunction

```
lemma cf-adjunction-of-counit-is-cf-adjunction:
 assumes category α ℭ
  and category α 𝔇
  and 𝔉 : ℭ ↦↦_Cα 𝔇
  and 𝔊 : 𝔇 ↦↦_Cα ℭ
  and ε : 𝔉 ∘_CF 𝔊 ↦_CF cf-id 𝔇 : 𝔇 ↦↦_Cα 𝔇
  and ⋀x. x ∈₀ 𝔇(|Obj|) ⟹ universal-arrow-fo 𝔉 x (𝔊(|ObjMap|)(|x|)) (ε(|NTMap|)(|x|))
```

shows *cf-adjunction-of-counit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \varepsilon : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
and ε_C (*cf-adjunction-of-counit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \varepsilon$) = ε
and $\mathcal{D}_\circ$ (*cf-adjunction-of-counit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \varepsilon(\!|AdjNT|\!)(\!|NTMap|\!)$) =
(*op-cat* $\mathfrak{C} \times_C \mathfrak{D}$)$(\!|Obj|\!)$
and $\bigwedge c\ d.\ [\![\ c \in_\circ \mathfrak{C}(\!|Obj|\!);\ d \in_\circ \mathfrak{D}(\!|Obj|\!)\ ]\!] \Longrightarrow$
cf-adjunction-of-counit $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \varepsilon(\!|AdjNT|\!)(\!|NTMap|\!)(\!|c,\ d|\!)_\bullet$ =
(*umap-fo* $\mathfrak{F}\ d\ (\mathfrak{G}(\!|ObjMap|\!)(\!|d|\!))\ (\varepsilon(\!|NTMap|\!)(\!|d|\!))\ c)^{-1}{}_{Set}$
proof–

interpret $\mathfrak{C}$: *category* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{D}$: *category* $\alpha\ \mathfrak{D}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{D}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*4*))
interpret ε: *is-ntcf* $\alpha\ \mathfrak{D}\ \mathfrak{D}\ \langle\mathfrak{F} \circ_{CF} \mathfrak{G}\rangle\ \langle$*cf-id* $\mathfrak{D}\rangle\ \varepsilon$ **by** (*rule assms*(*5*))

note *cf-adjunction-of-counit-def′* =
cf-adjunction-of-counit-def[**where** $\mathfrak{F}$=$\mathfrak{F}$, *unfolded* $\mathfrak{F}$.*cf-HomDom* $\mathfrak{F}$.*cf-HomCod*]

have *ua*:
universal-arrow-of (*op-cf* $\mathfrak{F}$) x (*op-cf* $\mathfrak{G}(\!|ObjMap|\!)(\!|x|\!)$) (*op-ntcf* $\varepsilon(\!|NTMap|\!)(\!|x|\!)$)
if $x \in_\circ$ *op-cat* $\mathfrak{D}(\!|Obj|\!)$ **for** x
using *that* **unfolding** *cat-op-simps* **by** (*rule assms*(*6*))

let *?aou* = $\langle$*cf-adjunction-of-unit* α (*op-cf* $\mathfrak{G}$) (*op-cf* $\mathfrak{F}$) (*op-ntcf* ε)$\rangle$
from
cf-adjunction-of-unit-is-cf-adjunction
[
OF
$\mathfrak{D}$.*category-op*
$\mathfrak{C}$.*category-op*
$\mathfrak{G}$.*is-functor-op*
$\mathfrak{F}$.*is-functor-op*
ε.*is-ntcf-op*[*unfolded cat-op-simps*]
ua,
simplified cf-adjunction-of-counit-def[*symmetric*]
]
have *aou*: *?aou* : *op-cf* $\mathfrak{G} \rightleftharpoons_{CF}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{D} \rightleftharpoons\rightleftharpoons_{C\alpha}$ *op-cat* $\mathfrak{C}$
and η-*aou*: η_C *?aou* = *op-ntcf* ε
by *auto*
interpret *aou*:
is-cf-adjunction $\alpha\ \langle$*op-cat* $\mathfrak{D}\rangle\ \langle$*op-cat* $\mathfrak{C}\rangle\ \langle$*op-cf* $\mathfrak{G}\rangle\ \langle$*op-cf* $\mathfrak{F}\rangle$ *?aou*
by (*rule aou*)
from η-*aou* **have**
op-ntcf (η_C *?aou*) = *op-ntcf* (*op-ntcf* ε)
by *simp*
then show ε_C (*cf-adjunction-of-counit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \varepsilon$) = ε
unfolding
ε.*ntcf-op-ntcf-op-ntcf*
is-cf-adjunction.op-ntcf-cf-adjunction-unit[*OF aou*]
cf-adjunction-of-counit-def′[*symmetric*]
by (*simp add*: *cat-op-simps*)
show *aoc-*ε: *cf-adjunction-of-counit* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \varepsilon : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
by
(
rule
is-cf-adjunction-op[
OF aou, *folded cf-adjunction-of-counit-def′*, *unfolded cat-op-simps*
]

```
    )
  interpret aoc-ε: is-cf-adjunction α ℭ 𝔇 𝔉 𝔊 ‹cf-adjunction-of-counit α 𝔉 𝔊 ε›
    by (rule aoc-ε)

  from aoc-ε.NT.is-ntcf-axioms show
    𝒟∘ (cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjNT|)(|NTMap|)) = (op-cat ℭ ×_C 𝔇)(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)

  show ⋀c d. [[ c ∈∘ ℭ(|Obj|); d ∈∘ 𝔇(|Obj|) ]] ⟹
    cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjNT|)(|NTMap|)(|c, d|)• =
      (umap-fo 𝔉 d (𝔊(|ObjMap|)(|d|)) (ε(|NTMap|)(|d|)) c)⁻¹_Set
  proof–
    fix c d assume prems: c ∈∘ ℭ(|Obj|) d ∈∘ 𝔇(|Obj|)
    from assms(1–4) prems have aou-dc:
      cf-adjunction-AdjNT-of-unit
        α (op-cf 𝔊) (op-cf 𝔉) (op-ntcf ε)(|NTMap|)(|d, c|)• =
        umap-fo 𝔉 d (𝔊(|ObjMap|)(|d|)) (ε(|NTMap|)(|d|)) c
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-op-simps adj-cs-simps cs-intro: cat-op-intros
        )
    from assms(1–4) aou prems have ufo-ε-dc:
      umap-fo 𝔉 d (𝔊(|ObjMap|)(|d|)) (ε(|NTMap|)(|d|)) c :
        Hom_{O.C α}op-cat ℭ(op-cf 𝔊−,−)(|ObjMap|)(|d, c|)• ↦_iso cat-Set α
        Hom_{O.C α}op-cat 𝔇(−,op-cf 𝔉−)(|ObjMap|)(|d, c|)•
      by
        (
          cs-concl cs-shallow
            cs-simp:
              aou-dc[symmetric] cf-adjunction-of-unit-components(3)[symmetric]
            cs-intro:
              is-iso-ntcf.iso-ntcf-is-iso-arr'
              adj-cs-intros
              cat-cs-intros
              cat-op-intros
              cat-prod-cs-intros
        )
    from
      assms(1–4)
      aoc-ε[unfolded cf-adjunction-of-counit-def']
      aou
      prems
      ufo-ε-dc
    show
      cf-adjunction-of-counit α 𝔉 𝔊 ε(|AdjNT|)(|NTMap|)(|c, d|)• =
        (umap-fo 𝔉 d (𝔊(|ObjMap|)(|d|)) (ε(|NTMap|)(|d|)) c)⁻¹_Set
      unfolding cf-adjunction-of-counit-def'
      by
        (
          cs-concl
            cs-simp: cat-op-simps adj-cs-simps cat-cs-simps cat-Set-cs-simps
            cs-intro: adj-cs-intros cat-cs-intros cat-prod-cs-intros
        )
  qed

qed
```

13.10 Construction of an adjunction from a functor and universal morphisms from functors to objects

The subsection presents the construction of an adjunction given a functor and a structured collection of universal morphisms from functors to objects. The content of this subsection follows the statement and the proof of Theorem 2-iv in Chapter IV-1 in [9].

13.10.1 Definition and elementary properties

definition *cf-ra-of-la* :: $(V \Rightarrow V) \Rightarrow V \Rightarrow V \Rightarrow V$
where $cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon = op\text{-}cf\ (cf\text{-}la\text{-}of\text{-}ra\ F\ (op\text{-}cf\ \mathfrak{F})\ \varepsilon)$

13.10.2 Object map

lemma *cf-ra-of-la-ObjMap-vsv*[*adj-cs-intros*]: $vsv\ (cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ObjMap|\!))$
unfolding *cf-ra-of-la-def op-cf-components* **by** (*auto intro*: *adj-cs-intros*)

lemma (**in** *is-functor*) *cf-ra-of-la-ObjMap-vdomain*:
$\mathcal{D}_\circ\ (cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ObjMap|\!)) = \mathfrak{B}(\!|Obj|\!)$
unfolding *cf-ra-of-la-def cf-la-of-ra-components cat-op-simps*
by (*simp add*: *cat-cs-simps*)

lemmas [*adj-cs-simps*] = *is-functor.cf-ra-of-la-ObjMap-vdomain*

lemma (**in** *is-functor*) *cf-ra-of-la-ObjMap-app*:
assumes $d \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ObjMap|\!)(\!|d|\!) = F\ d$
using *assms*
unfolding *cf-ra-of-la-def cf-la-of-ra-components cat-op-simps*
by (*simp add*: *cat-cs-simps*)

lemmas [*adj-cs-simps*] = *is-functor.cf-ra-of-la-ObjMap-app*

13.10.3 Arrow map

lemma *cf-ra-of-la-ArrMap-app-unique*:
assumes $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $f : a \mapsto_{\mathfrak{D}} b$
and $universal\text{-}arrow\text{-}fo\ \mathfrak{F}\ a\ (cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ObjMap|\!)(\!|a|\!))\ (\varepsilon(\!|a|\!))$
and $universal\text{-}arrow\text{-}fo\ \mathfrak{F}\ b\ (cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ObjMap|\!)(\!|b|\!))\ (\varepsilon(\!|b|\!))$
shows $cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ArrMap|\!)(\!|f|\!) : F\ a \mapsto_{\mathfrak{C}} F\ b$
and $f \circ_{A\mathfrak{D}} \varepsilon(\!|a|\!) =$
$umap\text{-}fo\ \mathfrak{F}\ b\ (F\ b)\ (\varepsilon(\!|b|\!))\ (F\ a)(\!|ArrVal|\!)(\!|cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ArrMap|\!)(\!|f|\!)|\!)$
and $\bigwedge f'.$
$[\![$
$f' : F\ a \mapsto_{\mathfrak{C}} F\ b;$
$f \circ_{A\mathfrak{D}} \varepsilon(\!|a|\!) = umap\text{-}fo\ \mathfrak{F}\ b\ (F\ b)\ (\varepsilon(\!|b|\!))\ (F\ a)(\!|ArrVal|\!)(\!|f'|\!)$
$]\!] \Longrightarrow cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\!|ArrMap|\!)(\!|f|\!) = f'$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
from *assms*(*2*) **have** *op-f*: $f : b \mapsto_{op\text{-}cat\ \mathfrak{D}} a$ **unfolding** *cat-op-simps* **by** *simp*
let *?lara* = $\langle cf\text{-}la\text{-}of\text{-}ra\ F\ (op\text{-}cf\ \mathfrak{F})\ \varepsilon\rangle$
have *lara-ObjMap-eq-op*: $?lara(\!|ObjMap|\!) = (op\text{-}cf\ ?lara(\!|ObjMap|\!))$
and *lara-ArrMap-eq-op*: $?lara(\!|ArrMap|\!) = (op\text{-}cf\ ?lara(\!|ArrMap|\!))$
unfolding *cat-op-simps* **by** *simp-all*
note *ua-η-a* = $\mathfrak{F}.universal\text{-}arrow\text{-}foD[OF\ assms(3)]$
and *ua-η-b* = $\mathfrak{F}.universal\text{-}arrow\text{-}foD[OF\ assms(4)]$
from *assms*(*1,2*) *ua-η-a*(*2*) **have** [*cat-op-simps*]:

```
    ε(|a|) ∘_{A op-cat 𝔇} f = f ∘_{A 𝔇} ε(|a|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-op-simps)
  show cf-ra-of-la F 𝔉 ε(|ArrMap|)(|f|) : F a ↦_ℭ F b
    and f ∘_{A 𝔇} ε(|a|) =
      umap-fo 𝔉 b (F b) (ε(|b|)) (F a)(|ArrVal|)(|cf-ra-of-la F 𝔉 ε(|ArrMap|)(|f|)|)
    and ⋀f'.
      ⟦
        f' : F a ↦_ℭ F b;
        f ∘_{A 𝔇} ε(|a|) = umap-fo 𝔉 b (F b) (ε(|b|)) (F a)(|ArrVal|)(|f'|)
      ⟧ ⟹ cf-ra-of-la F 𝔉 ε(|ArrMap|)(|f|) = f'
    by
      (
        intro
          cf-la-of-ra-ArrMap-app-unique
            [
              where η=ε and F=F,
                OF 𝔉.is-functor-op op-f,
                unfolded
                  𝔉.op-cf-universal-arrow-of
                  lara-ObjMap-eq-op
                  lara-ArrMap-eq-op,
                folded cf-ra-of-la-def,
                unfolded cat-op-simps, OF assms(4,3)
            ]
      )+
qed

lemma cf-ra-of-la-ArrMap-app-is-arr[adj-cs-intros]:
  assumes 𝔉 : ℭ ↦↦_{Cα} 𝔇
    and f : a ↦_𝔇 b
    and universal-arrow-fo 𝔉 a (cf-ra-of-la F 𝔉 ε(|ObjMap|)(|a|)) (ε(|a|))
    and universal-arrow-fo 𝔉 b (cf-ra-of-la F 𝔉 ε(|ObjMap|)(|b|)) (ε(|b|))
    and Fa = F a
    and Fb = F b
  shows cf-ra-of-la F 𝔉 ε(|ArrMap|)(|f|) : Fa ↦_ℭ Fb
  using assms(1–4) unfolding assms(5,6) by (rule cf-ra-of-la-ArrMap-app-unique)
```

13.10.4 An adjunction constructed from a functor and universal morphisms from functors to objects is an adjunction

```
lemma op-cf-cf-la-of-ra-op[cat-op-simps]:
  op-cf (cf-la-of-ra F (op-cf 𝔉) ε) = cf-ra-of-la F 𝔉 ε
  unfolding cf-ra-of-la-def by simp

lemma cf-ra-of-la-commute-op:
  assumes 𝔉 : ℭ ↦↦_{Cα} 𝔇
    and ⋀d. d ∈_∘ 𝔇(|Obj|) ⟹
      universal-arrow-fo 𝔉 d (cf-ra-of-la F 𝔉 ε(|ObjMap|)(|d|)) (ε(|d|))
    and ⋀d d' h. h : d ↦_𝔇 d' ⟹
      ε(|d'|) ∘_{A 𝔇} 𝔉(|ArrMap|)(|cf-ra-of-la F 𝔉 ε(|ArrMap|)(|h|)|) =
        h ∘_{A 𝔇} ε(|d|)
    and h : c' ↦_𝔇 c
  shows 𝔉(|ArrMap|)(|cf-ra-of-la F 𝔉 ε(|ArrMap|)(|h|)|) ∘_{A op-cat 𝔇} ε(|c|) =
    ε(|c'|) ∘_{A op-cat 𝔇} h
proof–
  interpret 𝔉: is-functor α ℭ 𝔇 𝔉 by (rule assms(1))
  from assms(4) have c': c' ∈_∘ 𝔇(|Obj|) and c: c ∈_∘ 𝔇(|Obj|) by auto
  note ua-η-c' = 𝔉.universal-arrow-foD[OF assms(2)[OF c']]
```

```
    and ua-η-c = 𝔉.universal-arrow-foD[OF assms(2)[OF c]]
  note rala-f = cf-ra-of-la-ArrMap-app-unique[
      OF assms(1) assms(4) assms(2)[OF c'] assms(2)[OF c]
      ]
  from assms(1) assms(4) ua-η-c'(2) ua-η-c(2) rala-f(1) show ?thesis
    by
      (
        cs-concl cs-shallow
          cs-simp: assms(3) cat-op-simps adj-cs-simps cat-cs-simps
          cs-intro: cat-cs-intros
      )
qed

lemma
  assumes 𝔉 : ℭ ↦↦_Cα 𝔇
    and ⋀d. d ∈₀ 𝔇(|Obj|) ⟹ F d ∈₀ ℭ(|Obj|)
    and ⋀d. d ∈₀ 𝔇(|Obj|) ⟹
      universal-arrow-fo 𝔉 d (cf-ra-of-la F 𝔉 ε(|ObjMap|)(|d|)) (ε(|d|))
    and ⋀d d' h. h : d ↦_𝔇 d' ⟹
      ε(|d'|) ∘_A𝔇 𝔉(|ArrMap|)(|cf-ra-of-la F 𝔉 ε(|ArrMap|)(|h|)|) =
        h ∘_A𝔇 ε(|d|)
  shows cf-ra-of-la-is-functor: cf-ra-of-la F 𝔉 ε : 𝔇 ↦↦_Cα ℭ
    and cf-la-of-ra-op-is-functor:
      cf-la-of-ra F (op-cf 𝔉) ε : op-cat 𝔇 ↦↦_Cα op-cat ℭ
proof–
  interpret 𝔉: is-functor α ℭ 𝔇 𝔉 by (rule assms(1))
  have 𝔉h-εc:
    𝔉(|ArrMap|)(|cf-ra-of-la F 𝔉 ε(|ArrMap|)(|h|)|) ∘_A op-cat 𝔇 ε(|c|) =
      ε(|c'|) ∘_A op-cat 𝔇 h
    if h : c' ↦_𝔇 c for c c' h
  proof–
    from that have c': c' ∈₀ 𝔇(|Obj|) and c: c ∈₀ 𝔇(|Obj|) by auto
    note ua-η-c' = 𝔉.universal-arrow-foD[OF assms(3)[OF c']]
      and ua-η-c = 𝔉.universal-arrow-foD[OF assms(3)[OF c]]
    note rala-f = cf-ra-of-la-ArrMap-app-unique[
        OF assms(1) that assms(3)[OF c'] assms(3)[OF c]
        ]
    from assms(1) that ua-η-c'(2) ua-η-c(2) rala-f(1) show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: assms(4) cat-op-simps adj-cs-simps cat-cs-simps
            cs-intro: cat-cs-intros
        )
  qed
  let ?lara = ‹cf-la-of-ra F (op-cf 𝔉) ε›
  have lara-ObjMap-eq-op: ?lara(|ObjMap|) = (op-cf ?lara(|ObjMap|))
    and lara-ArrMap-eq-op: ?lara(|ArrMap|) = (op-cf ?lara(|ArrMap|))
    by (simp-all add: cat-op-simps del: op-cf-cf-la-of-ra-op)
  show cf-la-of-ra F (op-cf 𝔉) ε : op-cat 𝔇 ↦↦_Cα op-cat ℭ
    by
      (
        intro cf-la-of-ra-is-functor
          [
            where F=F and η=ε,
            OF 𝔉.is-functor-op,
            unfolded cat-op-simps,
            OF assms(2),
```

```
        simplified,
        unfolded lara-ObjMap-eq-op lara-ArrMap-eq-op,
        folded cf-ra-of-la-def,
        OF assms(3) 𝔉h-εc,
        simplified
      ]
    )
  from
    is-functor.is-functor-op[
      OF this, unfolded cat-op-simps, folded cf-ra-of-la-def
      ]
  show cf-ra-of-la F 𝔉 ε : 𝔇 ↦↦_Cα ℭ.
qed

lemma cf-ra-of-la-is-ntcf:
  fixes F 𝔇 𝔉 𝔊 ε_m ε
  defines 𝔊 ≡ cf-ra-of-la F 𝔉 ε_m
    and ε ≡ [ε_m, 𝔉 ∘_CF 𝔊, cf-id 𝔇, 𝔇, 𝔇]_∘
  assumes 𝔉 : ℭ ↦↦_Cα 𝔇
    and ⋀d. d ∈_∘ 𝔇(|Obj|) ⟹ F d ∈_∘ ℭ(|Obj|)
    and ⋀d. d ∈_∘ 𝔇(|Obj|) ⟹
      universal-arrow-fo 𝔉 d (𝔊(|ObjMap|)(|d|)) (ε(|NTMap|)(|d|))
    and ⋀d d′ h. h : d ↦_𝔇 d′ ⟹
      ε(|NTMap|)(|d′|) ∘_A𝔇 𝔉(|ArrMap|)(|𝔊(|ArrMap|)(|h|)|) = h ∘_A𝔇 ε(|NTMap|)(|d|)
    and vsv (ε(|NTMap|))
    and 𝒟_∘ (ε(|NTMap|)) = 𝔇(|Obj|)
  shows ε : 𝔉 ∘_CF 𝔊 ↦_CF cf-id 𝔇 : 𝔇 ↦↦_Cα 𝔇
proof–
  interpret 𝔉: is-functor α ℭ 𝔇 𝔉 by (rule assms(3))
  have ε-components:
    ε(|NTMap|) = ε_m
    ε(|NTDom|) = 𝔉 ∘_CF 𝔊
    ε(|NTCod|) = cf-id 𝔇
    ε(|NTDGDom|) = 𝔇
    ε(|NTDGCod|) = 𝔇
    unfolding ε-def nt-field-simps by (simp-all add: nat-omega-simps)
  note 𝔊-def = 𝔊-def[folded ε-components(1)]
  interpret 𝔊: is-functor α 𝔇 ℭ 𝔊
    unfolding 𝔊-def
    by (auto intro: cf-ra-of-la-is-functor[OF assms(3–6)[unfolded 𝔊-def]])
  interpret op-ε: is-functor
    α ‹op-cat 𝔇› ‹op-cat ℭ› ‹cf-la-of-ra F (op-cf 𝔉) (ε(|NTMap|))›
    by
      (
        intro cf-la-of-ra-op-is-functor[
          where F=F and ε=‹ε(|NTMap|)›, OF assms(3–6)[unfolded 𝔊-def], simplified
          ]
      )
  interpret ε: vfsequence ε unfolding ε-def by simp
  have [cat-op-simps]: op-ntcf (op-ntcf ε) = ε
  proof(rule vsv-eqI)
    have dom-lhs: 𝒟_∘ (op-ntcf (op-ntcf ε)) = 5_ℕ
      unfolding op-ntcf-def by (simp add: nat-omega-simps)
    from assms(7) show 𝒟_∘ (op-ntcf (op-ntcf ε)) = 𝒟_∘ ε
      unfolding dom-lhs by (simp add: ε-def ε.vfsequence-vdomain nat-omega-simps)
    have sup:
      op-ntcf (op-ntcf ε)(|NTDom|) = ε(|NTDom|)
      op-ntcf (op-ntcf ε)(|NTCod|) = ε(|NTCod|)
```

```
    op-ntcf (op-ntcf ε)(|NTDGDom|) = ε(|NTDGDom|)
    op-ntcf (op-ntcf ε)(|NTDGCod|) = ε(|NTDGCod|)
    unfolding op-ntcf-components cat-op-simps ε-components
    by simp-all
  show a ∈∘ 𝒟∘ (op-ntcf (op-ntcf ε)) ⟹ op-ntcf (op-ntcf ε)(|a|) = ε(|a|) for a
    by (unfold dom-lhs, elim-in-numeral, fold nt-field-simps, unfold sup)
      (simp-all add: cat-op-simps)
qed (auto simp: op-ntcf-def)
let ?lara = ‹cf-la-of-ra F (op-cf 𝔉) (ε(|NTMap|))›
have lara-ObjMap-eq-op: ?lara(|ObjMap|) = (op-cf ?lara(|ObjMap|))
  and lara-ArrMap-eq-op: ?lara(|ArrMap|) = (op-cf ?lara(|ArrMap|))
  by (simp-all add: cat-op-simps del: op-cf-cf-la-of-ra-op)
have seq: vfsequence (op-ntcf ε) unfolding op-ntcf-def by auto
have card: vcard (op-ntcf ε) = 5ℕ
  unfolding op-ntcf-def by (simp add: nat-omega-simps)
have op-cf-NTCod: op-cf (ε(|NTCod|)) = cf-id (op-cat 𝔇)
  unfolding ε-components cat-op-simps by simp
from assms(3) have op-cf-NTDom:
  op-cf (ε(|NTDom|)) = op-cf 𝔉 ∘CF cf-la-of-ra F (op-cf 𝔉) (ε(|NTMap|))
  unfolding ε-components
  by
    (
      simp
        add: cat-op-simps 𝔊-def cf-ra-of-la-def ε-components(1)[symmetric]
        del: op-cf-cf-la-of-ra-op
    )
note cf-la-of-ra-is-ntcf
  [
    where F=F and ηm=‹ε(|NTMap|)›,
    OF is-functor.is-functor-op[OF assms(3)],
    unfolded cat-op-simps,
    OF assms(4),
    unfolded ε-components(1),
    folded op-cf-NTCod op-cf-NTDom[unfolded ε-components(1)] ε-components(1),
    folded op-ntcf-def[of ε, unfolded ε-components(4,5)],
    unfolded
      cat-op-simps
      lara-ObjMap-eq-op lara-ArrMap-eq-op cf-ra-of-la-def[symmetric],
    folded 𝔊-def,
    simplified,
    OF
      assms(5)
      cf-ra-of-la-commute-op[
        OF assms(3,5,6)[unfolded 𝔊-def], folded 𝔊-def
        ]
      assms(7,8),
    unfolded ε-components,
    simplified
  ]
from is-ntcf.is-ntcf-op[OF this, unfolded cat-op-simps 𝔊-def[symmetric]] show
  ε : 𝔉 ∘CF 𝔊 ↦CF cf-id 𝔇 : 𝔇 ↦↦Cα 𝔇.
qed

lemma cf-ra-of-la-is-counit:
  fixes F 𝔇 𝔉 𝔊 εm ε
  defines 𝔊 ≡ cf-ra-of-la F 𝔉 εm
    and ε ≡ [εm, 𝔉 ∘CF 𝔊, cf-id 𝔇, 𝔇, 𝔇]∘
  assumes category α ℭ
```

```
    and category α 𝔇
    and 𝔉 : ℭ ↦↦_Cα 𝔇
    and ⋀d. d ∈₀ 𝔇(|Obj|) ⟹ F d ∈₀ ℭ(|Obj|)
    and ⋀d. d ∈₀ 𝔇(|Obj|) ⟹
      universal-arrow-fo 𝔉 d (𝔊(|ObjMap|)(|d|)) (ε(|NTMap|)(|d|))
    and ⋀d d' h. h : d ↦_𝔇 d' ⟹
      ε(|NTMap|)(|d'|) ∘_A𝔇 𝔉(|ArrMap|)(|𝔊(|ArrMap|)(|h|)|) = h ∘_A𝔇 ε(|NTMap|)(|d|)
    and vfsequence ε
    and vsv (ε(|NTMap|))
    and 𝒟₀ (ε(|NTMap|)) = 𝔇(|Obj|)
  shows cf-adjunction-of-counit α 𝔉 𝔊 ε : 𝔉 ⇌_CF 𝔊 : ℭ ⇌⇌_Cα 𝔇
    and ε_C (cf-adjunction-of-counit α 𝔉 𝔊 ε) = ε
proof–
  have ε-components:
    ε(|NTMap|) = ε_m
    ε(|NTDom|) = 𝔉 ∘_CF 𝔊
    ε(|NTCod|) = cf-id 𝔇
    ε(|NTDGDom|) = 𝔇
    ε(|NTDGCod|) = 𝔇
    unfolding ε-def nt-field-simps by (simp-all add: nat-omega-simps)
  note 𝔊-def = 𝔊-def[folded ε-components(1)]
  note 𝔉 = cf-ra-of-la-is-functor[
    where F=F and ε=‹ε(|NTMap|)›, OF assms(5–8)[unfolded 𝔊-def], simplified
    ]
  note ε = cf-ra-of-la-is-ntcf
    [
      where F=F and ε_m=‹ε_m› and 𝔇=𝔇 and 𝔉=𝔉,
      folded 𝔊-def[unfolded ε-components(1)],
      folded ε-def,
      OF assms(5–8) assms(10,11),
      simplified
    ]
  show cf-adjunction-of-counit α 𝔉 𝔊 ε : 𝔉 ⇌_CF 𝔊 : ℭ ⇌⇌_Cα 𝔇
    and ε_C (cf-adjunction-of-counit α 𝔉 𝔊 ε) = ε
    by
      (
        intro cf-adjunction-of-counit-is-cf-adjunction
          [
            OF assms(3–5) 𝔉,
            folded 𝔊-def,
            OF ε assms(7)[folded 𝔊-def],
            simplified
          ]
      )+
qed
```

13.11 Construction of an adjunction from the counit-unit equations

The subsection presents the construction of an adjunction given two natural transformations satisfying counit-unit equations. The content of this subsection follows the statement and the proof of Theorem 2-v in Chapter IV-1 in [9].

```
lemma counit-unit-is-cf-adjunction:
  assumes category α ℭ
    and category α 𝔇
    and 𝔉 : ℭ ↦↦_Cα 𝔇
    and 𝔊 : 𝔇 ↦↦_Cα ℭ
    and η : cf-id ℭ ↦_CF 𝔊 ∘_CF 𝔉 : ℭ ↦↦_Cα ℭ
```

and $\varepsilon : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \mathit{cf\text{-}id}\ \mathfrak{D} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $(\mathfrak{G} \circ_{CF\text{-}NTCF} \varepsilon) \bullet_{NTCF} (\eta \circ_{NTCF\text{-}CF} \mathfrak{G}) = \mathit{ntcf\text{-}id}\ \mathfrak{G}$
and $(\varepsilon \circ_{NTCF\text{-}CF} \mathfrak{F}) \bullet_{NTCF} (\mathfrak{F} \circ_{CF\text{-}NTCF} \eta) = \mathit{ntcf\text{-}id}\ \mathfrak{F}$
shows $\mathit{cf\text{-}adjunction\text{-}of\text{-}unit}\ \alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
and $\eta_C\ (\mathit{cf\text{-}adjunction\text{-}of\text{-}unit}\ \alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta) = \eta$
and $\varepsilon_C\ (\mathit{cf\text{-}adjunction\text{-}of\text{-}unit}\ \alpha\ \mathfrak{F}\ \mathfrak{G}\ \eta) = \varepsilon$
proof–

interpret $\mathfrak{C}$: *category* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{D}$: *category* $\alpha\ \mathfrak{D}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{D}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*4*))
interpret η: *is-ntcf* $\alpha\ \mathfrak{C}\ \mathfrak{C}\ \langle \mathit{cf\text{-}id}\ \mathfrak{C}\rangle\ \langle \mathfrak{G} \circ_{CF} \mathfrak{F}\rangle\ \eta$ **by** (*rule assms*(*5*))
interpret ε: *is-ntcf* $\alpha\ \mathfrak{D}\ \mathfrak{D}\ \langle \mathfrak{F} \circ_{CF} \mathfrak{G}\rangle\ \langle \mathit{cf\text{-}id}\ \mathfrak{D}\rangle\ \varepsilon$ **by** (*rule assms*(*6*))

have $\mathfrak{G}\varepsilon x\text{-}\eta\mathfrak{G}x$[*cat-cs-simps*]:
$\mathfrak{G}(\lfloor \mathit{ArrMap}\rfloor)(\varepsilon(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{C}} \eta(\lfloor \mathit{NTMap}\rfloor)(\mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)) = \mathfrak{C}(\lfloor \mathit{CId}\rfloor)(\mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))$
if $x \in_{\circ} \mathfrak{D}(\lfloor \mathit{Obj}\rfloor)$ **for** x
proof–
from *assms*(*7*) **have**
$((\mathfrak{G} \circ_{CF\text{-}NTCF} \varepsilon) \bullet_{NTCF} (\eta \circ_{NTCF\text{-}CF} \mathfrak{G}))(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor) = \mathit{ntcf\text{-}id}\ \mathfrak{G}(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)$
by *simp*
from *this assms*(*1–6*) *that* **show**
$\mathfrak{G}(\lfloor \mathit{ArrMap}\rfloor)(\varepsilon(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{C}} \eta(\lfloor \mathit{NTMap}\rfloor)(\mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)) =$
$\mathfrak{C}(\lfloor \mathit{CId}\rfloor)(\mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed
have [*cat-cs-simps*]:
$\mathfrak{G}(\lfloor \mathit{ArrMap}\rfloor)(\varepsilon(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{C}} (\eta(\lfloor \mathit{NTMap}\rfloor)(\mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{C}} f) =$
$\mathfrak{C}(\lfloor \mathit{CId}\rfloor)(\mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{C}} f$
if $x \in_{\circ} \mathfrak{D}(\lfloor \mathit{Obj}\rfloor)$ **and** $f : a \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)$ **for** $x\ f\ a$
using *assms*(*1–6*) *that*
by (*intro* $\mathfrak{C}$.*cat-assoc-helper*)
(*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)+

have [*cat-cs-simps*]:
$\varepsilon(\lfloor \mathit{NTMap}\rfloor)(\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{D}} \mathfrak{F}(\lfloor \mathit{ArrMap}\rfloor)(\eta(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)) = \mathfrak{D}(\lfloor \mathit{CId}\rfloor)(\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))$
if $x \in_{\circ} \mathfrak{C}(\lfloor \mathit{Obj}\rfloor)$ **for** x
proof–
from *assms*(*8*) **have**
$((\varepsilon \circ_{NTCF\text{-}CF} \mathfrak{F}) \bullet_{NTCF} (\mathfrak{F} \circ_{CF\text{-}NTCF} \eta))(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor) = \mathit{ntcf\text{-}id}\ \mathfrak{F}(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)$
by *simp*
from *this assms*(*1–6*) *that* **show**
$\varepsilon(\lfloor \mathit{NTMap}\rfloor)(\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor)) \circ_{A\mathfrak{D}} \mathfrak{F}(\lfloor \mathit{ArrMap}\rfloor)(\eta(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor)) = \mathfrak{D}(\lfloor \mathit{CId}\rfloor)(\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

have $\mathit{ua}\text{-}\mathfrak{F}x\text{-}\eta x$: $\mathit{universal\text{-}arrow\text{-}of}\ \mathfrak{G}\ x\ (\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))\ (\eta(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor))$
if $x \in_{\circ} \mathfrak{C}(\lfloor \mathit{Obj}\rfloor)$ **for** x
proof(*intro is-functor.universal-arrow-ofI*)
from *assms*(*3*) *that* **show** $\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor) \in_{\circ} \mathfrak{D}(\lfloor \mathit{Obj}\rfloor)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms*(*3–6*) *that* **show** $\eta(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor) : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
fix $r'\ u'$ **assume** *prems'*: $r' \in_{\circ} \mathfrak{D}(\lfloor \mathit{Obj}\rfloor)\ u' : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor r'\rfloor)$
show $\exists! f'.$
$f' : \mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor) \mapsto_{\mathfrak{D}} r' \wedge$
$u' = \mathit{umap\text{-}of}\ \mathfrak{G}\ x\ (\mathfrak{F}(\lfloor \mathit{ObjMap}\rfloor)(\lfloor x\rfloor))\ (\eta(\lfloor \mathit{NTMap}\rfloor)(\lfloor x\rfloor))\ r'(\lfloor \mathit{ArrVal}\rfloor)(\lfloor f'\rfloor)$

```
  proof(intro ex1I conjI; (elim conjE)?)
    from assms(3–6) that prems′ show
      ε(⁅NTMap⁆)(⁅r′⁆) ∘_A 𝔇 𝔉(⁅ArrMap⁆)(⁅u′⁆) : 𝔉(⁅ObjMap⁆)(⁅x⁆) ↦_𝔇 r′
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3–6) prems′ have 𝔊𝔉u′:
      (𝔊 ∘_CF 𝔉)(⁅ArrMap⁆)(⁅u′⁆) = 𝔊(⁅ArrMap⁆)(⁅𝔉(⁅ArrMap⁆)(⁅u′⁆)⁆)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    note [cat-cs-simps] =
      η.ntcf-Comp-commute[symmetric, OF prems′(2), unfolded 𝔊𝔉u′]
    from assms(3–6) that prems′ show
      u′ =
        umap-of 𝔊 x (𝔉(⁅ObjMap⁆)(⁅x⁆)) (η(⁅NTMap⁆)(⁅x⁆)) r′(⁅ArrVal⁆)(⁅ε(⁅NTMap⁆)(⁅r′⁆) ∘_A𝔇
        𝔉(⁅ArrMap⁆)(⁅u′⁆)⁆)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
        )
    fix f′ assume prems′′:
      f′ : 𝔉(⁅ObjMap⁆)(⁅x⁆) ↦_𝔇 r′
      u′ = umap-of 𝔊 x (𝔉(⁅ObjMap⁆)(⁅x⁆)) (η(⁅NTMap⁆)(⁅x⁆)) r′(⁅ArrVal⁆)(⁅f′⁆)
    from prems′′(2,1) assms(3–6) that have u′-def:
      u′ = 𝔊(⁅ArrMap⁆)(⁅f′⁆) ∘_A𝔈 η(⁅NTMap⁆)(⁅x⁆)
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
        )
    from
      ε.ntcf-Comp-commute[OF prems′′(1)]
      assms(3–6)
      prems′′(1)
    have [cat-cs-simps]:
      ε(⁅NTMap⁆)(⁅r′⁆) ∘_A𝔇 𝔉(⁅ArrMap⁆)(⁅𝔊(⁅ArrMap⁆)(⁅f′⁆)⁆) =
        f′ ∘_A𝔇 ε(⁅NTMap⁆)(⁅𝔉(⁅ObjMap⁆)(⁅x⁆)⁆)
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    have [cat-cs-simps]:
      ε(⁅NTMap⁆)(⁅r′⁆) ∘_A𝔇 (𝔉(⁅ArrMap⁆)(⁅𝔊(⁅ArrMap⁆)(⁅f′⁆)⁆) ∘_A𝔇 f) =
        (f′ ∘_A𝔇 ε(⁅NTMap⁆)(⁅𝔉(⁅ObjMap⁆)(⁅x⁆)⁆)) ∘_A𝔇 f
      if f : a ↦_𝔇 𝔉(⁅ObjMap⁆)(⁅𝔊(⁅ObjMap⁆)(⁅𝔉(⁅ObjMap⁆)(⁅x⁆)⁆)⁆) for f a
      using assms(1–6) prems′′(1) prems′ that
      by (intro 𝔇.cat-assoc-helper)
        (
          cs-concl
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
        )+
    from prems′′(2,1) assms(3–6) that show
      f′ = ε(⁅NTMap⁆)(⁅r′⁆) ∘_A𝔇 𝔉(⁅ArrMap⁆)(⁅u′⁆)
      unfolding u′-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
        )
  qed
qed (auto intro: cat-cs-intros)

show aou: cf-adjunction-of-unit α 𝔉 𝔊 η : 𝔉 ⇌_CF 𝔊 : 𝔈 ⇌⇌_Cα 𝔇
```

```
    by (intro cf-adjunction-of-unit-is-cf-adjunction ua-𝔉x-ηx assms(1–5))
  from ℭ.category-axioms 𝔇.category-axioms show
    η_C (cf-adjunction-of-unit α 𝔉 𝔊 η) = η
    by
      (
        cs-concl cs-shallow
          cs-intro: cf-adjunction-of-unit-is-cf-adjunction assms(1–5) ua-𝔉x-ηx
      )

  interpret aou: is-cf-adjunction α ℭ 𝔇 𝔉 𝔊 ‹cf-adjunction-of-unit α 𝔉 𝔊 η›
    by (rule aou)

  show ε_C (cf-adjunction-of-unit α 𝔉 𝔊 η) = ε
  proof(rule ntcf-eqI)
    show ε-η: ε_C (cf-adjunction-of-unit α 𝔉 𝔊 η) :
      𝔉 ∘_CF 𝔊 ↦_CF cf-id 𝔇 : 𝔇 ↦↦_Cα 𝔇
      by (rule aou.cf-adjunction-counit-is-ntcf)
    from assms(1–6) ε-η have dom-lhs:
      𝒟∘ (ε_C (cf-adjunction-of-unit α 𝔉 𝔊 η)(|NTMap|)) = 𝔇(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from assms(1–6) ε-η have dom-rhs: 𝒟∘ (ε(|NTMap|)) = 𝔇(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    show ε_C (cf-adjunction-of-unit α 𝔉 𝔊 η)(|NTMap|) = ε(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈∘ 𝔇(|Obj|)
      with aou.is-cf-adjunction-axioms assms(1–6) show
        ε_C (cf-adjunction-of-unit α 𝔉 𝔊 η)(|NTMap|)(|a|) = ε(|NTMap|)(|a|)
        by
          (
            cs-concl
              cs-intro:
                cat-arrow-cs-intros
                cat-op-intros
                cat-cs-intros
                cat-prod-cs-intros
              cs-simp:
                aou.cf-adj-umap-of-unit'[symmetric]
                cat-Set-the-inverse[symmetric]
                adj-cs-simps cat-cs-simps cat-op-simps
          )
    qed (auto simp: adj-cs-intros)
  qed (auto simp: assms)

qed

lemma counit-unit-cf-adjunction-of-counit-is-cf-adjunction:
  assumes category α ℭ
    and category α 𝔇
    and 𝔉 : ℭ ↦↦_Cα 𝔇
    and 𝔊 : 𝔇 ↦↦_Cα ℭ
    and η : cf-id ℭ ↦_CF 𝔊 ∘_CF 𝔉 : ℭ ↦↦_Cα ℭ
    and ε : 𝔉 ∘_CF 𝔊 ↦_CF cf-id 𝔇 : 𝔇 ↦↦_Cα 𝔇
    and (𝔊 ∘_CF–NTCF ε) ⋆_NTCF (η ∘_NTCF–CF 𝔊) = ntcf-id 𝔊
    and (ε ∘_NTCF–CF 𝔉) ⋆_NTCF (𝔉 ∘_CF–NTCF η) = ntcf-id 𝔉
  shows cf-adjunction-of-counit α 𝔉 𝔊 ε : 𝔉 ⇌_CF 𝔊 : ℭ ⇌⇌_Cα 𝔇
    and η_C (cf-adjunction-of-counit α 𝔉 𝔊 ε) = η
    and ε_C (cf-adjunction-of-counit α 𝔉 𝔊 ε) = ε
proof–
```

interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{D}$: *category* α $\mathfrak{D}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{D}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*4*))
interpret η: *is-ntcf* α $\mathfrak{C}$ $\mathfrak{C}$ ‹*cf-id* $\mathfrak{C}$› ‹$\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$› η **by** (*rule assms*(*5*))
interpret ε: *is-ntcf* α $\mathfrak{D}$ $\mathfrak{D}$ ‹$\mathfrak{F}$ $\circ_{CF}$ $\mathfrak{G}$› ‹*cf-id* $\mathfrak{D}$› ε **by** (*rule assms*(*6*))

have *unit-op*: *cf-adjunction-of-unit* α (*op-cf* $\mathfrak{G}$) (*op-cf* $\mathfrak{F}$) (*op-ntcf* ε) :
  op-cf $\mathfrak{G}$ $\rightleftharpoons_{CF}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{D}$ $\rightleftharpoons\rightleftharpoons_{C\alpha}$ *op-cat* $\mathfrak{C}$
  by (*rule counit-unit-is-cf-adjunction*(*1*)[**where** ε=‹*op-ntcf* η›])
    (
      cs-concl
        cs-simp:
          cat-op-simps cat-cs-simps
          $\mathfrak{G}$.*cf-ntcf-id-op-cf*
          $\mathfrak{F}$.*cf-ntcf-id-op-cf*
          op-ntcf-ntcf-vcomp[*symmetric*]
          op-ntcf-ntcf-cf-comp[*symmetric*]
          op-ntcf-cf-ntcf-comp[*symmetric*]
          assms(*7*,*8*)
        cs-intro: *cat-op-intros cat-cs-intros*
)+
then show *aou*: *cf-adjunction-of-counit* α $\mathfrak{F}$ $\mathfrak{G}$ ε : $\mathfrak{F}$ $\rightleftharpoons_{CF}$ $\mathfrak{G}$: $\mathfrak{C}$ $\rightleftharpoons\rightleftharpoons_{C\alpha}$ $\mathfrak{D}$
  unfolding *cf-adjunction-of-counit-def*
  by
    (
      subst $\mathfrak{F}$.*cf-op-cf-op-cf*[*symmetric*],
      subst $\mathfrak{G}$.*cf-op-cf-op-cf*[*symmetric*],
      subst $\mathfrak{C}$.*cat-op-cat-op-cat*[*symmetric*],
      subst $\mathfrak{D}$.*cat-op-cat-op-cat*[*symmetric*],
      rule is-cf-adjunction-op
)

interpret *aou*: *is-cf-adjunction* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ ‹*cf-adjunction-of-counit* α $\mathfrak{F}$ $\mathfrak{G}$ ε›
  by (*rule aou*)

show η_C (*cf-adjunction-of-counit* α $\mathfrak{F}$ $\mathfrak{G}$ ε) = η
  unfolding *cf-adjunction-of-counit-def*
  by
    (
      cs-concl-step is-cf-adjunction.op-ntcf-cf-adjunction-counit[*symmetric*],
      rule unit-op,
      cs-concl-step counit-unit-is-cf-adjunction(*3*)[**where** ε=‹*op-ntcf* η›],
      insert $\mathfrak{C}$.*category-op* $\mathfrak{D}$.*category-op*,
      rule $\mathfrak{D}$.*category-op*, *rule* $\mathfrak{C}$.*category-op*
)
    (
      cs-concl
        cs-simp:
          cat-op-simps cat-cs-simps
          $\mathfrak{G}$.*cf-ntcf-id-op-cf*
          $\mathfrak{F}$.*cf-ntcf-id-op-cf*
          op-ntcf-ntcf-vcomp[*symmetric*]
          op-ntcf-ntcf-cf-comp[*symmetric*]
          op-ntcf-cf-ntcf-comp[*symmetric*]
          assms(*7*,*8*)
        cs-intro: *cat-op-intros cat-cs-intros*

```
    )+

  show ε_C (cf-adjunction-of-counit α 𝔉 𝔊 ε) = ε
    unfolding cf-adjunction-of-counit-def
    by
      (
        cs-concl-step is-cf-adjunction.op-ntcf-cf-adjunction-unit[symmetric],
        rule unit-op,
        cs-concl-step counit-unit-is-cf-adjunction(2)[where ε=‹op-ntcf η›],
        insert ℭ.category-op 𝔇.category-op,
        rule 𝔇.category-op, rule ℭ.category-op
      )
      (
        cs-concl
          cs-simp:
            cat-op-simps cat-cs-simps
            𝔊.cf-ntcf-id-op-cf
            𝔉.cf-ntcf-id-op-cf
            op-ntcf-ntcf-vcomp[symmetric]
            op-ntcf-ntcf-cf-comp[symmetric]
            op-ntcf-cf-ntcf-comp[symmetric]
            assms(7,8)
          cs-intro: cat-op-intros cat-cs-intros
      )+

qed
```

13.12 Adjoints are unique up to isomorphism

The content of the following subsection is based predominantly on the statement and the proof of Corollary 1 in Chapter IV-1 in [9]. However, similar results can also be found in section 4 in [14] and in subsection 2.1 in [4].

13.12.1 Definitions and elementary properties

```
definition cf-adj-LR-iso :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where cf-adj-LR-iso ℭ 𝔇 𝔊 𝔉 Φ 𝔉' Ψ =
    [
      (
        λx∈₀ℭ(|Obj|). THE f'.
        let
          η = η_C Φ;
          η' = η_C Ψ;
          𝔉x = 𝔉(|ObjMap|)(|x|);
          𝔉'x = 𝔉'(|ObjMap|)(|x|)
        in
          f' : 𝔉x ↦_𝔇 𝔉'x ∧
          η'(|NTMap|)(|x|) = umap-of 𝔊 x (𝔉x) (η(|NTMap|)(|x|)) (𝔉'x)(|ArrVal|)(|f'|)
      ),
      𝔉,
      𝔉',
      ℭ,
      𝔇
    ]₀

definition cf-adj-RL-iso :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ =
```

```
  [
    (
      λx∈₀𝔇(|Obj|). THE f'.
      let
        ε = ε_C Φ;
        ε' = ε_C Ψ;
        𝔊x = 𝔊(|ObjMap|)(|x|);
        𝔊'x = 𝔊'(|ObjMap|)(|x|)
      in
        f' : 𝔊'x ↦_ℭ 𝔊x ∧
        ε'(|NTMap|)(|x|) = umap-fo 𝔉 x 𝔊x (ε(|NTMap|)(|x|)) 𝔊'x(|ArrVal|)(|f'|)
    ),
    𝔊',
    𝔊,
    𝔇,
    ℭ
  ]∘
```

Components.

```
lemma cf-adj-LR-iso-components:
  shows cf-adj-LR-iso ℭ 𝔇 𝔊 𝔉 Φ 𝔉' Ψ(|NTMap|) =
    (
      λx∈₀ℭ(|Obj|). THE f'.
      let
        η = η_C Φ;
        η' = η_C Ψ;
        𝔉x = 𝔉(|ObjMap|)(|x|);
        𝔉'x = 𝔉'(|ObjMap|)(|x|)
      in
        f' : 𝔉x ↦_𝔇 𝔉'x ∧
        η'(|NTMap|)(|x|) = umap-of 𝔊 x 𝔉x (η(|NTMap|)(|x|)) 𝔉'x(|ArrVal|)(|f'|)
    )
    and [adj-cs-simps]: cf-adj-LR-iso ℭ 𝔇 𝔊 𝔉 Φ 𝔉' Ψ(|NTDom|) = 𝔉
    and [adj-cs-simps]: cf-adj-LR-iso ℭ 𝔇 𝔊 𝔉 Φ 𝔉' Ψ(|NTCod|) = 𝔉'
    and [adj-cs-simps]: cf-adj-LR-iso ℭ 𝔇 𝔊 𝔉 Φ 𝔉' Ψ(|NTDGDom|) = ℭ
    and [adj-cs-simps]: cf-adj-LR-iso ℭ 𝔇 𝔊 𝔉 Φ 𝔉' Ψ(|NTDGCod|) = 𝔇
  unfolding cf-adj-LR-iso-def nt-field-simps
  by (simp-all add: nat-omega-simps)

lemma cf-adj-RL-iso-components:
  shows cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ(|NTMap|) =
    (
      λx∈₀𝔇(|Obj|). THE f'.
      let
        ε = ε_C Φ;
        ε' = ε_C Ψ;
        𝔊x = 𝔊(|ObjMap|)(|x|);
        𝔊'x = 𝔊'(|ObjMap|)(|x|)
      in
        f' : 𝔊'x ↦_ℭ 𝔊x ∧
        ε'(|NTMap|)(|x|) = umap-fo 𝔉 x 𝔊x (ε(|NTMap|)(|x|)) 𝔊'x(|ArrVal|)(|f'|)
    )
    and [adj-cs-simps]: cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ(|NTDom|) = 𝔊'
    and [adj-cs-simps]: cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ(|NTCod|) = 𝔊
    and [adj-cs-simps]: cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ(|NTDGDom|) = 𝔇
    and [adj-cs-simps]: cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ(|NTDGCod|) = ℭ
  unfolding cf-adj-RL-iso-def nt-field-simps
  by (simp-all add: nat-omega-simps)
```

13.12.2 Natural transformation map

lemma *cf-adj-LR-iso-vsv*[*adj-cs-intros*]:
vsv (*cf-adj-LR-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{G}$ $\mathfrak{F}$ Φ $\mathfrak{F}'$ Ψ(|*NTMap*|))
unfolding *cf-adj-LR-iso-components* **by** *simp*

lemma *cf-adj-RL-iso-vsv*[*adj-cs-intros*]:
vsv (*cf-adj-RL-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ Φ $\mathfrak{G}'$ Ψ(|*NTMap*|))
unfolding *cf-adj-RL-iso-components* **by** *simp*

lemma *cf-adj-LR-iso-vdomain*[*adj-cs-simps*]:
$\mathcal{D}_\circ$ (*cf-adj-LR-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{G}$ $\mathfrak{F}$ Φ $\mathfrak{F}'$ Ψ(|*NTMap*|)) = $\mathfrak{C}$(|*Obj*|)
unfolding *cf-adj-LR-iso-components* **by** *simp*

lemma *cf-adj-RL-iso-vdomain*[*adj-cs-simps*]:
$\mathcal{D}_\circ$ (*cf-adj-RL-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ Φ $\mathfrak{G}'$ Ψ(|*NTMap*|)) = $\mathfrak{D}$(|*Obj*|)
unfolding *cf-adj-RL-iso-components* **by** *simp*

lemma *cf-adj-LR-iso-app*:
fixes $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{G}$ $\mathfrak{F}$ Φ $\mathfrak{F}'$ Ψ
assumes $x \in_\circ \mathfrak{C}$(|*Obj*|)
defines $\mathfrak{F}x \equiv \mathfrak{F}$(|*ObjMap*|)(|$x$|)
and $\mathfrak{F}'x \equiv \mathfrak{F}'$(|*ObjMap*|)(|$x$|)
and $\eta \equiv \eta_C\ \Phi$
and $\eta' \equiv \eta_C\ \Psi$
shows *cf-adj-LR-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{G}$ $\mathfrak{F}$ Φ $\mathfrak{F}'$ Ψ(|*NTMap*|)(|x|) =
(
THE f'.
$f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x \wedge$
η'(|*NTMap*|)(|x|) = *umap-of* $\mathfrak{G}$ x $\mathfrak{F}x$ (η(|*NTMap*|)(|x|)) $\mathfrak{F}'x$(|*ArrVal*|)(|f'|)
)
using *assms*(*1*) **unfolding** *cf-adj-LR-iso-components assms*(*2–5*) **by** *simp meson*

lemma *cf-adj-RL-iso-app*:
fixes $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ Φ $\mathfrak{G}'$ Ψ
assumes $x \in_\circ \mathfrak{D}$(|*Obj*|)
defines $\mathfrak{G}x \equiv \mathfrak{G}$(|*ObjMap*|)(|$x$|)
and $\mathfrak{G}'x \equiv \mathfrak{G}'$(|*ObjMap*|)(|$x$|)
and $\varepsilon \equiv \varepsilon_C\ \Phi$
and $\varepsilon' \equiv \varepsilon_C\ \Psi$
shows *cf-adj-RL-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$ Φ $\mathfrak{G}'$ Ψ(|*NTMap*|)(|x|) =
(
THE f'.
$f' : \mathfrak{G}'x \mapsto_{\mathfrak{C}} \mathfrak{G}x \wedge$
ε'(|*NTMap*|)(|x|) = *umap-fo* $\mathfrak{F}$ x $\mathfrak{G}x$ (ε(|*NTMap*|)(|x|)) $\mathfrak{G}'x$(|*ArrVal*|)(|f'|)
)
using *assms*(*1*) **unfolding** *cf-adj-RL-iso-components assms*(*2–5*) *Let-def* **by** *simp*

lemma *cf-adj-LR-iso-app-unique*:
fixes $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{G}$ $\mathfrak{F}$ Φ $\mathfrak{F}'$ Ψ
assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
and $\Psi : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
and $x \in_\circ \mathfrak{C}$(|*Obj*|)
defines $\mathfrak{F}x \equiv \mathfrak{F}$(|*ObjMap*|)(|$x$|)
and $\mathfrak{F}'x \equiv \mathfrak{F}'$(|*ObjMap*|)(|$x$|)
and $\eta \equiv \eta_C\ \Phi$
and $\eta' \equiv \eta_C\ \Psi$
and $f \equiv$ *cf-adj-LR-iso* $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{G}$ $\mathfrak{F}$ Φ $\mathfrak{F}'$ Ψ(|*NTMap*|)(|x|)

shows
$\exists!f'.$
$f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x \wedge$
$\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f' \rfloor)$
$f : \mathfrak{F}x \mapsto_{iso\mathfrak{D}} \mathfrak{F}'x$
$\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$
proof–
interpret Φ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}\ \Phi$ **by** (*rule assms*(*1*))
interpret Ψ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}'\ \mathfrak{G}\ \Psi$ **by** (*rule assms*(*2*))
note $\mathfrak{F}a\text{-}\eta$ =
is-cf-adjunction.cf-adjunction-unit-component-is-ua-of[
OF assms(*1*) *assms*(*3*), *folded assms*(*4–8*)
]
note $\mathfrak{F}'a\text{-}\eta$ =
is-cf-adjunction.cf-adjunction-unit-component-is-ua-of[
OF assms(*2*) *assms*(*3*), *folded assms*(*4–8*)
]
from
is-functor.cf-universal-arrow-of-unique[
OF Φ*.RL.is-functor-axioms* $\mathfrak{F}a\text{-}\eta\ \mathfrak{F}'a\text{-}\eta$, *folded assms*(*4–8*)
]
obtain f'
where f': $f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x$
and η'*-def*:
$\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f' \rfloor)$
and *unique-f'*:
$[\![$
$f'' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x;$
$\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f'' \rfloor)$
$]\!] \Longrightarrow f'' = f'$
for f''
by *metis*
show *unique-f'*: $\exists!f'.$
$f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x \wedge$
$\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f' \rfloor)$
by
(
rule is-functor.cf-universal-arrow-of-unique[
OF Φ*.RL.is-functor-axioms* $\mathfrak{F}a\text{-}\eta\ \mathfrak{F}'a\text{-}\eta$, *folded assms*(*4–8*)
]
)
from
theD
[
OF unique-f' cf-adj-LR-iso-app[
OF assms(*3*), *of* $\mathfrak{D}\ \mathfrak{G}\ \mathfrak{F}\ \Phi\ \mathfrak{F}'\ \Psi$, *folded assms*(*4–8*)
]
]
have f: $f : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x$
and η': $\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$
by *simp-all*
show $\eta'(\lfloor NTMap \rfloor)(\lfloor x \rfloor) = umap\text{-}of\ \mathfrak{G}\ x\ \mathfrak{F}x\ (\eta(\lfloor NTMap \rfloor)(\lfloor x \rfloor))\ \mathfrak{F}'x(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$ **by** (*rule* η')
show $f : \mathfrak{F}x \mapsto_{iso\mathfrak{D}} \mathfrak{F}'x$
by
(
rule
is-functor.cf-universal-arrow-of-is-iso-arr[
OF Φ*.RL.is-functor-axioms* $\mathfrak{F}a\text{-}\eta\ \mathfrak{F}'a\text{-}\eta\ f\ \eta'$

]
)
qed

13.12.3 Main results

lemma *cf-adj-LR-iso-is-iso-functor*:
— See Corollary 1 in Chapter IV-1 in [9].
assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$ **and** $\Psi : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
shows $\exists!\vartheta.\ \vartheta : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D} \wedge \eta_C\ \Psi = (\mathfrak{G} \circ_{CF-NTCF} \vartheta) \cdot_{NTCF} \eta_C\ \Phi$
and *cf-adj-LR-iso* $\mathfrak{C}\ \mathfrak{D}\ \mathfrak{G}\ \mathfrak{F}\ \Phi\ \mathfrak{F}'\ \Psi : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{F}' : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\eta_C\ \Psi = (\mathfrak{G} \circ_{CF-NTCF}$ *cf-adj-LR-iso* $\mathfrak{C}\ \mathfrak{D}\ \mathfrak{G}\ \mathfrak{F}\ \Phi\ \mathfrak{F}'\ \Psi) \cdot_{NTCF} \eta_C\ \Phi$
proof–

interpret Φ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}\ \Phi$ **by** (*rule assms*(*1*))
interpret Ψ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}'\ \mathfrak{G}\ \Psi$ **by** (*rule assms*(*2*))

let *?η* = ‹$\eta_C\ \Phi$›
let *?η′* = ‹$\eta_C\ \Psi$›
let *?ΦΨ* = ‹*cf-adj-LR-iso* $\mathfrak{C}\ \mathfrak{D}\ \mathfrak{G}\ \mathfrak{F}\ \Phi\ \mathfrak{F}'\ \Psi$›

show $\mathfrak{F}'\Psi$: *?ΦΨ* : $\mathfrak{F} \mapsto_{CF.iso} \mathfrak{F}' : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof(*intro is-iso-ntcfI is-ntcfI′*)

show *vfsequence ?ΦΨ* **unfolding** *cf-adj-LR-iso-def* **by** *auto*
show *vcard ?ΦΨ* = $5_{\mathbb{N}}$
unfolding *cf-adj-LR-iso-def* **by** (*simp add*: *nat-omega-simps*)
show *?ΦΨ*(⟦*NTMap*⟧)(⟦*a*⟧) : $\mathfrak{F}$(⟦*ObjMap*⟧)(⟦*a*⟧) $\mapsto_{\mathfrak{D}}$ $\mathfrak{F}'$(⟦*ObjMap*⟧)(⟦*a*⟧)
if *a* $\in_\circ$ $\mathfrak{C}$(⟦*Obj*⟧) **for** *a*
using *cf-adj-LR-iso-app-unique*(*2*)[*OF assms that*] **by** *auto*

show *?ΦΨ*(⟦*NTMap*⟧)(⟦*b*⟧) $\circ_{A\mathfrak{D}}$ $\mathfrak{F}$(⟦*ArrMap*⟧)(⟦*f*⟧) = $\mathfrak{F}'$(⟦*ArrMap*⟧)(⟦*f*⟧) $\circ_{A\mathfrak{D}}$ *?ΦΨ*(⟦*NTMap*⟧)(⟦*a*⟧)
if *f* : *a* $\mapsto_{\mathfrak{C}}$ *b* **for** *a b f*
proof–

from *that* **have** *a*: *a* $\in_\circ$ $\mathfrak{C}$(⟦*Obj*⟧) **and** *b*: *b* $\in_\circ$ $\mathfrak{C}$(⟦*Obj*⟧) **by** *auto*
note *unique-a* = *cf-adj-LR-iso-app-unique*[*OF assms a*]
note *unique-b* = *cf-adj-LR-iso-app-unique*[*OF assms b*]

from *unique-a*(*2*) **have** *a-is-arr*:
?ΦΨ(⟦*NTMap*⟧)(⟦*a*⟧) : $\mathfrak{F}$(⟦*ObjMap*⟧)(⟦*a*⟧) $\mapsto_{\mathfrak{D}}$ $\mathfrak{F}'$(⟦*ObjMap*⟧)(⟦*a*⟧)
by *auto*
from *unique-b*(*2*) **have** *b-is-arr*:
?ΦΨ(⟦*NTMap*⟧)(⟦*b*⟧) : $\mathfrak{F}$(⟦*ObjMap*⟧)(⟦*b*⟧) $\mapsto_{\mathfrak{D}}$ $\mathfrak{F}'$(⟦*ObjMap*⟧)(⟦*b*⟧)
by *auto*

interpret *η*: *is-ntcf* $\alpha\ \mathfrak{C}\ \mathfrak{C}$ ‹*cf-id* $\mathfrak{C}$› ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› *?η*
by (*rule* Φ.*cf-adjunction-unit-is-ntcf*)
interpret *η′*: *is-ntcf* $\alpha\ \mathfrak{C}\ \mathfrak{C}$ ‹*cf-id* $\mathfrak{C}$› ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}'$› *?η′*
by (*rule* Ψ.*cf-adjunction-unit-is-ntcf*)

from *unique-a*(*3*) *a-is-arr a b* **have** *η′-a-def*:
?η′(⟦*NTMap*⟧)(⟦*a*⟧) = $\mathfrak{G}$(⟦*ArrMap*⟧)(⟦*?ΦΨ*(⟦*NTMap*⟧)(⟦*a*⟧)⟧) $\circ_{A\mathfrak{C}}$ *?η*(⟦*NTMap*⟧)(⟦*a*⟧)
by
(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros cat-cs-intros*
)

from *unique-b*(*3*) *b-is-arr a b* **have** η'*-b-def*:
$?\eta'(\lfloor NTMap \rfloor)(\lfloor b \rfloor) = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \rfloor) \circ_{A\mathfrak{C}} ?\eta(\lfloor NTMap \rfloor)(\lfloor b \rfloor)$
by
(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros cat-cs-intros*
)

from *that a b a-is-arr* **have**
$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor) \circ_{A\mathfrak{C}}$
$(\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor a \rfloor) \rfloor) \circ_{A\mathfrak{C}} ?\eta(\lfloor NTMap \rfloor)(\lfloor a \rfloor)) =$
$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor) \circ_{A\mathfrak{C}} ?\eta'(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* η'*-a-def* **cs-intro**: *cat-cs-intros*
)
also from η'*.ntcf-Comp-commute*[*OF that*, *symmetric*] *that a b* **have**
$\ldots = ?\eta'(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{C}} f$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
also from *that a b b-is-arr* **have**
$\ldots = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \rfloor) \circ_{A\mathfrak{C}}$
$(?\eta(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{C}} f)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* η'*-b-def* **cs-intro**: *cat-cs-intros*
)
also from *that* **have**
$\ldots = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \rfloor) \circ_{A\mathfrak{C}}$
$((\mathfrak{G} \circ_{CF} \mathfrak{F})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} ?\eta(\lfloor NTMap \rfloor)(\lfloor a \rfloor))$
unfolding η*.ntcf-Comp-commute*[*OF that*, *symmetric*]
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* η'*-b-def* **cs-intro**: *cat-cs-intros*
)
also from *that b-is-arr* **have**
$\ldots = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \rfloor) \circ_{A\mathfrak{C}}$
$(\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor) \circ_{A\mathfrak{C}} ?\eta(\lfloor NTMap \rfloor)(\lfloor a \rfloor))$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
finally have [*cat-cs-simps*]:
$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor a \rfloor) \rfloor) \circ_{A\mathfrak{C}}$
$?\eta(\lfloor NTMap \rfloor)(\lfloor a \rfloor)) =$
$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor ?\Phi\Psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \rfloor) \circ_{A\mathfrak{C}}$
$(\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor) \circ_{A\mathfrak{C}} ?\eta(\lfloor NTMap \rfloor)(\lfloor a \rfloor))$
by *simp*

note *unique-f-a* = *is-functor.universal-arrow-ofD*
[
OF
Φ*.RL.is-functor-axioms*
Φ*.cf-adjunction-unit-component-is-ua-of*[*OF a*]
]

from *that a b a-is-arr b-is-arr* **have** $\mathfrak{G}\mathfrak{F}$*f-*η*a*:
$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor) \circ_{A\mathfrak{C}} ?\eta'(\lfloor NTMap \rfloor)(\lfloor a \rfloor) :$
$a \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}'(\lfloor ObjMap \rfloor)(\lfloor b \rfloor) \rfloor)$

```
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  from b have 𝔉'b: 𝔉'(|ObjMap|)(|b|) ∈∘ 𝔇(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  from unique-f-a(3)[OF 𝔉'b 𝔊𝔉f-ηa] obtain f'
    where f': f' : 𝔉(|ObjMap|)(|a|) ↦𝔇 𝔉'(|ObjMap|)(|b|)
      and ηa: 𝔊(|ArrMap|)(|𝔉'(|ArrMap|)(|f|)|) ∘A𝔈 ?η'(|NTMap|)(|a|) =
      umap-of 𝔊 a (𝔉(|ObjMap|)(|a|)) (?η(|NTMap|)(|a|)) (𝔉'(|ObjMap|)(|b|))(|ArrVal|)(|f'|)
      and unique-f':
        ⟦
          f'' : 𝔉(|ObjMap|)(|a|) ↦𝔇 𝔉'(|ObjMap|)(|b|);
          𝔊(|ArrMap|)(|𝔉'(|ArrMap|)(|f|)|) ∘A𝔈 ?η'(|NTMap|)(|a|) =
            umap-of 𝔊 a (𝔉(|ObjMap|)(|a|)) (?η(|NTMap|)(|a|)) (𝔉'(|ObjMap|)(|b|))(|ArrVal|)(|f''|)
        ⟧ ⟹ f'' = f'
    for f''
    by metis
  have ?ΦΨ(|NTMap|)(|b|) ∘A𝔇 𝔉(|ArrMap|)(|f|) = f'
    by (rule unique-f', insert a b a-is-arr b-is-arr that)
      (
        cs-concl cs-shallow
          cs-simp: η'-a-def cat-cs-simps cs-intro: cat-cs-intros
      )
  moreover have 𝔉'(|ArrMap|)(|f|) ∘A𝔇 ?ΦΨ(|NTMap|)(|a|) = f'
    by (rule unique-f', insert a b a-is-arr b-is-arr that)
      (
        cs-concl cs-shallow
          cs-simp: η'-a-def cat-cs-simps cs-intro: cat-cs-intros
      )
  ultimately show ?thesis by simp
qed

qed
(
  auto
    intro: cat-cs-intros adj-cs-intros
    simp: adj-cs-simps cf-adj-LR-iso-app-unique(2)[OF assms]
)

interpret 𝔉'Ψ: is-iso-ntcf α 𝔈 𝔇 𝔉 𝔉' ‹?ΦΨ› by (rule 𝔉'Ψ)

show η'-def: ?η' = 𝔊 ∘CF−NTCF ?ΦΨ ⊙NTCF ηC Φ
proof(rule ntcf-eqI)
  have dom-lhs: 𝒟∘ (?η'(|NTMap|)) = 𝔈(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: adj-cs-intros)
  have dom-rhs: 𝒟∘ ((𝔊 ∘CF−NTCF ?ΦΨ ⊙NTCF ηC Φ)(|NTMap|)) = 𝔈(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros
      )
  show ?η'(|NTMap|) = (𝔊 ∘CF−NTCF ?ΦΨ ⊙NTCF ηC Φ)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume prems: a ∈∘ 𝔈(|Obj|)
    note unique-a = cf-adj-LR-iso-app-unique[OF assms prems]
    from unique-a(2) have a-is-arr:
      ?ΦΨ(|NTMap|)(|a|) : 𝔉(|ObjMap|)(|a|) ↦𝔇 𝔉'(|ObjMap|)(|a|)
      by auto
```

```
    interpret η: is-ntcf α 𝔈 𝔈 ‹cf-id 𝔈› ‹𝔊 ∘CF 𝔉› ?η
      by (rule Φ.cf-adjunction-unit-is-ntcf)
    from unique-a(3) a-is-arr prems have η'-a-def:
      ?η'(|NTMap|)(|a|) = 𝔊(|ArrMap|)(|?ΦΨ(|NTMap|)(|a|)|) ∘A𝔈 ηC Φ(|NTMap|)(|a|)
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros
        )
    from prems a-is-arr show
      ?η'(|NTMap|)(|a|) = (𝔊 ∘CF-NTCF ?ΦΨ ·NTCF ?η)(|NTMap|)(|a|)
      by
        (
          cs-concl cs-shallow
            cs-simp: η'-a-def cat-cs-simps cs-intro: cat-cs-intros
        )
  qed (auto intro: cat-cs-intros adj-cs-intros)
qed
  (
    cs-concl cs-shallow
      cs-simp: cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros
  )+

show ∃!ϑ. ϑ : 𝔉 ↦CF 𝔉' : 𝔈 ↦↦Cα 𝔇 ∧ ?η' = (𝔊 ∘CF-NTCF ϑ) ·NTCF ?η
proof(intro ex1I conjI; (elim conjE)?)
  from 𝔉'Ψ show ?ΦΨ : 𝔉 ↦CF 𝔉' : 𝔈 ↦↦Cα 𝔇 by auto
  show ?η' = 𝔊 ∘CF-NTCF ?ΦΨ ·NTCF ηC Φ by (rule η'-def)
  fix ϑ assume prems:
    ϑ : 𝔉 ↦CF 𝔉' : 𝔈 ↦↦Cα 𝔇
    ?η' = 𝔊 ∘CF-NTCF ϑ ·NTCF ηC Φ
  interpret ϑ: is-ntcf α 𝔈 𝔇 𝔉 𝔉' ϑ by (rule prems(1))
  from prems have η'-a:
    ?η'(|NTMap|)(|a|) = (𝔊 ∘CF-NTCF ϑ ·NTCF ηC Φ)(|NTMap|)(|a|)
    for a
    by simp
  have η'a: ηC Ψ(|NTMap|)(|a|) =
    𝔊(|ArrMap|)(|ϑ(|NTMap|)(|a|)|) ∘A𝔈 ηC Φ(|NTMap|)(|a|)
    if a ∈∘ 𝔈(|Obj|) for a
    using η'-a[where a=a] that
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros
      )
  show ϑ = ?ΦΨ
  proof(rule ntcf-eqI)
    have dom-lhs: 𝒟∘ (ϑ(|NTMap|)) = 𝔈(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    have dom-rhs: 𝒟∘ (?ΦΨ(|NTMap|)) = 𝔈(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    show ϑ(|NTMap|) = ?ΦΨ(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume prems': a ∈∘ 𝔈(|Obj|)
      let ?uof = ‹umap-of 𝔊 a (𝔉(|ObjMap|)(|a|)) (?η(|NTMap|)(|a|)) (𝔉'(|ObjMap|)(|a|))›
      from cf-adj-LR-iso-app-unique[OF assms prems'] obtain f'
        where f': f' : 𝔉(|ObjMap|)(|a|) ↦𝔇 𝔉'(|ObjMap|)(|a|)
          and η-def: ?η'(|NTMap|)(|a|) = ?uof(|ArrVal|)(|f'|)
          and unique-f': ⋀f''.
```

```
        ⟦
          f'' : ℱ(⁅ObjMap⁆)(⁅a⁆) ↦_𝔇 ℱ'(⁅ObjMap⁆)(⁅a⁆);
          ?η'(⁅NTMap⁆)(⁅a⁆) = ?uof(⁅ArrVal⁆)(⁅f''⁆)
        ⟧ ⟹ f'' = f'
      by metis
    from prems' have ϑa: ϑ(⁅NTMap⁆)(⁅a⁆) : ℱ(⁅ObjMap⁆)(⁅a⁆) ↦_𝔇 ℱ'(⁅ObjMap⁆)(⁅a⁆)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from η-def f' prems' have
      η_C Ψ(⁅NTMap⁆)(⁅a⁆) = 𝔊(⁅ArrMap⁆)(⁅f'⁆) ∘_A𝔈 η_C Φ(⁅NTMap⁆)(⁅a⁆)
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros
        )
    from prems' have η_C Ψ(⁅NTMap⁆)(⁅a⁆) = ?uof(⁅ArrVal⁆)(⁅ϑ(⁅NTMap⁆)(⁅a⁆)⁆)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps η'a[OF prems']
            cs-intro: adj-cs-intros cat-cs-intros
        )
    from unique-f'[OF ϑa this] have ϑa: ϑ(⁅NTMap⁆)(⁅a⁆) = f'.
    from prems' have Ψa:
      ?ΦΨ(⁅NTMap⁆)(⁅a⁆) : ℱ(⁅ObjMap⁆)(⁅a⁆) ↦_𝔇 ℱ'(⁅ObjMap⁆)(⁅a⁆)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from prems' have η_C Ψ(⁅NTMap⁆)(⁅a⁆) = ?uof(⁅ArrVal⁆)(⁅?ΦΨ(⁅NTMap⁆)(⁅a⁆)⁆)
      by
        (
          cs-concl
            cs-simp: cf-adj-LR-iso-app-unique(3)[OF assms] cat-cs-simps
            cs-intro: adj-cs-intros cat-cs-intros
        )
    from unique-f'[OF Ψa this] have ℱ'Ψ-def: ?ΦΨ(⁅NTMap⁆)(⁅a⁆) = f'.
    show ϑ(⁅NTMap⁆)(⁅a⁆) = ?ΦΨ(⁅NTMap⁆)(⁅a⁆) unfolding ϑa ℱ'Ψ-def ..
  qed auto
 qed (cs-concl cs-shallow cs-intro: cat-cs-intros)+
qed

qed

lemma op-ntcf-cf-adj-RL-iso[cat-op-simps]:
  assumes Φ : ℱ ⇌_CF 𝔊 : 𝔈 ⇌⇌_Cα 𝔇
    and Ψ : ℱ ⇌_CF 𝔊' : 𝔈 ⇌⇌_Cα 𝔇
  defines op-𝔇 ≡ op-cat 𝔇
    and op-𝔈 ≡ op-cat 𝔈
    and op-ℱ ≡ op-cf ℱ
    and op-𝔊 ≡ op-cf 𝔊
    and op-Φ ≡ op-cf-adj Φ
    and op-𝔊' ≡ op-cf 𝔊'
    and op-Ψ ≡ op-cf-adj Ψ
  shows
    op-ntcf (cf-adj-RL-iso 𝔈 𝔇 ℱ 𝔊 Φ 𝔊' Ψ) =
      cf-adj-LR-iso op-𝔇 op-𝔈 op-ℱ op-𝔊 op-Φ op-𝔊' op-Ψ
proof–
  interpret Φ: is-cf-adjunction α 𝔈 𝔇 ℱ 𝔊 Φ by (rule assms(1))
  interpret Ψ: is-cf-adjunction α 𝔈 𝔇 ℱ 𝔊' Ψ by (rule assms(2))
  interpret ε: is-ntcf α 𝔇 𝔇 ‹ℱ ∘_CF 𝔊› ‹cf-id 𝔇› ‹ε_C Φ›
    by (rule Φ.cf-adjunction-counit-is-ntcf)
```

```
  have dom-lhs: 𝒟∘ (op-ntcf (cf-adj-RL-iso 𝔈 𝔇 𝔉 𝔊 Φ 𝔊' Ψ)) = 5ℕ
    unfolding op-ntcf-def by (simp add: nat-omega-simps)
  show ?thesis
  proof(rule vsv-eqI, unfold dom-lhs)
    fix a assume prems: a ∈∘ 5ℕ
    then have a ∈∘ 5ℕ unfolding dom-lhs by simp
    then show
      op-ntcf (cf-adj-RL-iso 𝔈 𝔇 𝔉 𝔊 Φ 𝔊' Ψ)(⌊a⌋) =
        cf-adj-LR-iso op-𝔇 op-𝔈 op-𝔉 op-𝔊 op-Φ op-𝔊' op-Ψ(⌊a⌋)
      by
        (
          elim-in-numeral,
          fold nt-field-simps,
          unfold
            cf-adj-LR-iso-components
            op-ntcf-components
            cf-adj-RL-iso-components
            Let-def
            Φ.cf-adjunction-unit-NTMap-op
            Ψ.cf-adjunction-unit-NTMap-op
            assms(3–9)
            cat-op-simps
        )
        simp-all
  qed (auto simp: op-ntcf-def cf-adj-LR-iso-def nat-omega-simps)
qed

lemma op-ntcf-cf-adj-LR-iso[cat-op-simps]:
  assumes Φ : 𝔉 ⇌CF 𝔊 : 𝔈 ⇌⇌Cα 𝔇 and Ψ : 𝔉' ⇌CF 𝔊 : 𝔈 ⇌⇌Cα 𝔇
  defines op-𝔇 ≡ op-cat 𝔇
    and op-𝔈 ≡ op-cat 𝔈
    and op-𝔉 ≡ op-cf 𝔉
    and op-𝔊 ≡ op-cf 𝔊
    and op-Φ ≡ op-cf-adj Φ
    and op-𝔉' ≡ op-cf 𝔉'
    and op-Ψ ≡ op-cf-adj Ψ
  shows
    op-ntcf (cf-adj-LR-iso 𝔈 𝔇 𝔊 𝔉 Φ 𝔉' Ψ) =
      cf-adj-RL-iso op-𝔇 op-𝔈 op-𝔊 op-𝔉 op-Φ op-𝔉' op-Ψ
proof–
  interpret Φ: is-cf-adjunction α 𝔈 𝔇 𝔉 𝔊 Φ by (rule assms(1))
  interpret Ψ: is-cf-adjunction α 𝔈 𝔇 𝔉' 𝔊 Ψ by (rule assms(2))
  interpret ε: is-ntcf α 𝔇 𝔇 ‹𝔉 ∘CF 𝔊› ‹cf-id 𝔇› ‹εC Φ›
    by (rule Φ.cf-adjunction-counit-is-ntcf)
  have dom-lhs: 𝒟∘ (op-ntcf (cf-adj-LR-iso 𝔈 𝔇 𝔊 𝔉 Φ 𝔉' Ψ)) = 5ℕ
    unfolding op-ntcf-def by (simp add: nat-omega-simps)
  show ?thesis
  proof(rule vsv-eqI, unfold dom-lhs)
    fix a assume prems: a ∈∘ 5ℕ
    then show
      op-ntcf (cf-adj-LR-iso 𝔈 𝔇 𝔊 𝔉 Φ 𝔉' Ψ)(⌊a⌋) =
        cf-adj-RL-iso op-𝔇 op-𝔈 op-𝔊 op-𝔉 op-Φ op-𝔉' op-Ψ(⌊a⌋)
      by
        (
          elim-in-numeral,
          use nothing in
            ‹
              fold nt-field-simps,
```

```
        unfold
          cf-adj-LR-iso-components
          op-ntcf-components
          cf-adj-RL-iso-components
          Let-def
          Φ.op-ntcf-cf-adjunction-unit[symmetric]
          Ψ.op-ntcf-cf-adjunction-unit[symmetric]
          assms(3–9)
          cat-op-simps
      ›
    )
    simp-all
  qed (auto simp: op-ntcf-def cf-adj-RL-iso-def nat-omega-simps)
qed

lemma cf-adj-RL-iso-app-unique:
  fixes ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ
  assumes Φ : 𝔉 ⇌CF 𝔊 : ℭ ⇌⇌Cα 𝔇
    and Ψ : 𝔉 ⇌CF 𝔊' : ℭ ⇌⇌Cα 𝔇
    and x ∈∘ 𝔇(|Obj|)
  defines 𝔊x ≡ 𝔊(|ObjMap|)(|x|)
    and 𝔊'x ≡ 𝔊'(|ObjMap|)(|x|)
    and ε ≡ εC Φ
    and ε' ≡ εC Ψ
    and f ≡ cf-adj-RL-iso ℭ 𝔇 𝔉 𝔊 Φ 𝔊' Ψ(|NTMap|)(|x|)
  shows
    ∃!f'.
      f' : 𝔊'x ↦ℭ 𝔊x ∧
      ε'(|NTMap|)(|x|) = umap-fo 𝔉 x 𝔊x (ε(|NTMap|)(|x|)) 𝔊'x(|ArrVal|)(|f'|)
    f : 𝔊'x ↦iso.ℭ 𝔊x
    ε'(|NTMap|)(|x|) = umap-fo 𝔉 x 𝔊x (ε(|NTMap|)(|x|)) 𝔊'x(|ArrVal|)(|f|)
proof–
  interpret Φ: is-cf-adjunction α ℭ 𝔇 𝔉 𝔊 Φ by (rule assms(1))
  interpret Ψ: is-cf-adjunction α ℭ 𝔇 𝔉 𝔊' Ψ by (rule assms(2))
  interpret ε: is-ntcf α 𝔇 𝔇 ‹𝔉 ∘CF 𝔊› ‹cf-id 𝔇› ‹εC Φ›
    by (rule Φ.cf-adjunction-counit-is-ntcf)
  show
    ∃!f'.
      f' : 𝔊'x ↦ℭ 𝔊x ∧
      ε'(|NTMap|)(|x|) = umap-fo 𝔉 x 𝔊x (ε(|NTMap|)(|x|)) 𝔊'x(|ArrVal|)(|f'|)
    f : 𝔊'x ↦iso.ℭ 𝔊x
    ε'(|NTMap|)(|x|) = umap-fo 𝔉 x 𝔊x (ε(|NTMap|)(|x|)) 𝔊'x(|ArrVal|)(|f|)
    by
      (
        intro cf-adj-LR-iso-app-unique
          [
            OF Φ.is-cf-adjunction-op Ψ.is-cf-adjunction-op,
            unfolded cat-op-simps,
            OF assms(3),
            unfolded Ψ.cf-adjunction-unit-NTMap-op,
            folded Φ.op-ntcf-cf-adjunction-counit,
            folded op-ntcf-cf-adj-RL-iso[OF assms(1,2)],
            unfolded cat-op-simps,
            folded assms(4–8)
          ]
      )+
qed
```

lemma *cf-adj-RL-iso-is-iso-functor*:
— See Corollary 1 in Chapter IV-1 in [9].
assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$ **and** $\Psi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G}' : \mathfrak{C} \rightleftharpoons\rightleftharpoons_{C\alpha} \mathfrak{D}$
shows $\exists!\vartheta.$
$\vartheta : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C} \wedge$
$\varepsilon_C\ \Psi = \varepsilon_C\ \Phi \bullet_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \vartheta)$
and *cf-adj-RL-iso* $\mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}\ \Phi\ \mathfrak{G}'\ \Psi : \mathfrak{G}' \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\varepsilon_C\ \Psi =$
$\varepsilon_C\ \Phi \bullet_{NTCF} (\mathfrak{F} \circ_{CF-NTCF}$ *cf-adj-RL-iso* $\mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}\ \Phi\ \mathfrak{G}'\ \Psi)$
proof–
interpret Φ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}\ \Phi$ **by** (*rule assms*(*1*))
interpret Ψ: *is-cf-adjunction* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}'\ \Psi$ **by** (*rule assms*(*2*))
interpret ε: *is-ntcf* $\alpha\ \mathfrak{D}\ \mathfrak{D}\ \langle\mathfrak{F} \circ_{CF} \mathfrak{G}\rangle\ \langle$*cf-id* $\mathfrak{D}\rangle\ \langle\varepsilon_C\ \Phi\rangle$
by (*rule* Φ.*cf-adjunction-counit-is-ntcf*)
note *cf-adj-LR-iso-is-iso-functor-op* = *cf-adj-LR-iso-is-iso-functor*
[
OF Φ.*is-cf-adjunction-op* Ψ.*is-cf-adjunction-op*,
folded
Φ.*op-ntcf-cf-adjunction-counit*
Ψ.*op-ntcf-cf-adjunction-counit*
op-ntcf-cf-adj-RL-iso[*OF assms*]
]
from *cf-adj-LR-iso-is-iso-functor-op*(*1*) **obtain** ϑ
where ϑ: ϑ : *op-cf* $\mathfrak{G} \mapsto_{CF}$ *op-cf* $\mathfrak{G}'$: *op-cat* $\mathfrak{D} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
and *op-ntcf-ε-def*: *op-ntcf* $(\varepsilon_C\ \Psi) =$
op-cf $\mathfrak{F} \circ_{CF-NTCF} \vartheta \bullet_{NTCF}$ *op-ntcf* $(\varepsilon_C\ \Phi)$
and *unique-*ϑ':
$[\![$
ϑ' : *op-cf* $\mathfrak{G} \mapsto_{CF}$ *op-cf* $\mathfrak{G}'$: *op-cat* $\mathfrak{D} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$;
op-ntcf $(\varepsilon_C\ \Psi) =$ *op-cf* $\mathfrak{F} \circ_{CF-NTCF} \vartheta' \bullet_{NTCF}$ *op-ntcf* $(\varepsilon_C\ \Phi)$
$]\!] \Longrightarrow \vartheta' = \vartheta$
for ϑ'
by *metis*
interpret ϑ: *is-ntcf* $\alpha\ \langle$*op-cat* $\mathfrak{D}\rangle\ \langle$*op-cat* $\mathfrak{C}\rangle\ \langle$*op-cf* $\mathfrak{G}\rangle\ \langle$*op-cf* $\mathfrak{G}'\rangle\ \vartheta$
by (*rule* ϑ)
show $\exists!\vartheta.\ \vartheta : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C} \wedge \varepsilon_C\ \Psi = \varepsilon_C\ \Phi \bullet_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \vartheta)$
proof(*intro ex1I conjI*; (*elim conjE*)*?*)
show *op-*ϑ: *op-ntcf* $\vartheta : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*rule* ϑ.*is-ntcf-op*[*unfolded cat-op-simps*])
from *op-ntcf-ε-def* **have**
op-ntcf (*op-ntcf* $(\varepsilon_C\ \Psi)) =$
op-ntcf (*op-cf* $\mathfrak{F} \circ_{CF-NTCF} \vartheta \bullet_{NTCF}$ *op-ntcf* $(\varepsilon_C\ \Phi))$
by *simp*
then show *ε-def*: $\varepsilon_C\ \Psi = \varepsilon_C\ \Phi \bullet_{NTCF} (\mathfrak{F} \circ_{CF-NTCF}$ *op-ntcf* $\vartheta)$
by
(
cs-prems **cs-shallow**
cs-simp: *cat-op-simps*
cs-intro: *adj-cs-intros cat-cs-intros cat-op-intros*
)
fix ϑ' **assume** *prems*:
ϑ' : $\mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
$\varepsilon_C\ \Psi = \varepsilon_C\ \Phi \bullet_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \vartheta')$
interpret ϑ': *is-ntcf* $\alpha\ \mathfrak{D}\ \mathfrak{C}\ \mathfrak{G}'\ \mathfrak{G}\ \vartheta'$ **by** (*rule prems*(*1*))
have *op-ntcf* $(\varepsilon_C\ \Psi) =$ *op-cf* $\mathfrak{F} \circ_{CF-NTCF}$ *op-ntcf* $\vartheta' \bullet_{NTCF}$ *op-ntcf* $(\varepsilon_C\ \Phi)$
by
(
cs-concl **cs-shallow**

```
          cs-simp:
            prems(2)
            op-ntcf-cf-ntcf-comp[symmetric]
            op-ntcf-ntcf-vcomp[symmetric]
          cs-intro: cat-cs-intros
      )
    from unique-ϑ'[OF ϑ'.is-ntcf-op this, symmetric] have
      op-ntcf ϑ = op-ntcf (op-ntcf ϑ')
      by simp
    then show ϑ' = op-ntcf ϑ
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cat-op-simps) simp
  qed
  from is-iso-ntcf.is-iso-ntcf-op[OF cf-adj-LR-iso-is-iso-functor-op(2)] show
    cf-adj-RL-iso ℭ 𝔇 ℱ 𝒢 Φ 𝒢' Ψ : 𝒢' ↦_{CF.iso} 𝒢 : 𝔇 ↦↦_{Cα} ℭ
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-op-simps cs-intro: adj-cs-intros cat-op-intros
      )
  from cf-adj-LR-iso-is-iso-functor-op(3) have
    op-ntcf (op-ntcf (ε_C Ψ)) =
      op-ntcf
        (
          op-cf ℱ ∘_{CF-NTCF} op-ntcf (cf-adj-RL-iso ℭ 𝔇 ℱ 𝒢 Φ 𝒢' Ψ) ·_{NTCF}
          op-ntcf (ε_C Φ)
        )
    by simp
  from
    this
    cf-adj-LR-iso-is-iso-functor-op(2)[
      unfolded op-ntcf-cf-adj-RL-iso[OF assms]
      ]
  show ε_C Ψ = ε_C Φ ·_{NTCF} (ℱ ∘_{CF-NTCF} cf-adj-RL-iso ℭ 𝔇 ℱ 𝒢 Φ 𝒢' Ψ)
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-op-simps cat-op-simps
          cs-intro: ntcf-cs-intros adj-cs-intros cat-cs-intros cat-op-intros
      )
qed
```

13.13 Further properties of the adjoint functors

```
lemma (in is-cf-adjunction) cf-adj-exp-cf-cat:
  — See Proposition 4.4.6 in [14].
  assumes 𝒵 β and α ∈₀ β and category α 𝔍
  shows
    cf-adjunction-of-unit
      β
      (exp-cf-cat α ℱ 𝔍)
      (exp-cf-cat α 𝒢 𝔍)
      (exp-ntcf-cat α (η_C Φ) 𝔍) :
      exp-cf-cat α ℱ 𝔍 ⇌_{CF} exp-cf-cat α 𝒢 𝔍 :
      cat-FUNCT α 𝔍 ℭ ⇌⇌_{Cβ} cat-FUNCT α 𝔍 𝔇
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret 𝔍: category α 𝔍 by (rule assms(3))
  show ?thesis
```

proof
(
rule counit-unit-is-cf-adjunction(*1*)[
where ε = ‹*exp-ntcf-cat* α (ε_C Φ) $\mathfrak{J}$›
]
)
from *assms* **show** *exp-ntcf-cat* α (η_C Φ) $\mathfrak{J}$:
cf-id (*cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}$) $\mapsto_{CF}$ *exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{J}$ $\circ_{CF}$ *exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{J}$:
cat-FUNCT α $\mathfrak{J}$ $\mathfrak{C}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}$
by
(
cs-concl
cs-simp:
cat-cs-simps cat-FUNCT-cs-simps
exp-cf-cat-cf-id-cat[*symmetric*] *exp-cf-cat-cf-comp*[*symmetric*]
cs-intro:
cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros adj-cs-intros
)
from *assms* **show**
exp-ntcf-cat α (ε_C Φ) $\mathfrak{J}$:
exp-cf-cat α $\mathfrak{F}$ $\mathfrak{J}$ $\circ_{CF}$ *exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{J}$ $\mapsto_{CF}$ *cf-id* (*cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{D}$) :
cat-FUNCT α $\mathfrak{J}$ $\mathfrak{D}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{D}$
by
(
cs-concl
cs-simp:
cat-cs-simps
cat-FUNCT-cs-simps
exp-cf-cat-cf-id-cat[*symmetric*]
exp-cf-cat-cf-comp[*symmetric*]
cs-intro:
cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros adj-cs-intros
)
note [*symmetric*, *cat-cs-simps*] =
ntcf-id-exp-cf-cat
exp-ntcf-cat-ntcf-vcomp
exp-ntcf-cat-ntcf-cf-comp
exp-ntcf-cat-cf-ntcf-comp
from *assms* **show**
(*exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{J}$ $\circ_{CF-NTCF}$ *exp-ntcf-cat* α (ε_C Φ) $\mathfrak{J}$) $\cdot_{NTCF}$
(*exp-ntcf-cat* α (η_C Φ) $\mathfrak{J}$ $\circ_{NTCF-CF}$ *exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{J}$) =
ntcf-id (*exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{J}$)
by
(
cs-concl **cs-shallow**
cs-simp: *adj-cs-simps cat-cs-simps*
cs-intro: *adj-cs-intros cat-cs-intros*
)
from *assms* **show**
exp-ntcf-cat α (ε_C Φ) $\mathfrak{J}$ $\circ_{NTCF-CF}$ *exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{J}$ $\cdot_{NTCF}$
(*exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{J}$ $\circ_{CF-NTCF}$ *exp-ntcf-cat* α (η_C Φ) $\mathfrak{J}$) =
ntcf-id (*exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{J}$)
by
(
cs-concl **cs-shallow**
cs-simp: *adj-cs-simps cat-cs-simps*
cs-intro: *adj-cs-intros cat-cs-intros*
)

```
  qed
    (
      use assms in
        ‹
          cs-concl
            cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
        ›
    )+
qed

lemma (in is-cf-adjunction) cf-adj-exp-cf-cat-exp-cf-cat:
  — See Proposition 4.4.6 in [14].
  assumes 𝒵 β and α ∈₀ β and category α 𝔄
  shows
    cf-adjunction-of-unit
      β
      (exp-cat-cf α 𝔄 𝔊)
      (exp-cat-cf α 𝔄 𝔉)
      (exp-cat-ntcf α 𝔄 (η_C Φ)) :
      exp-cat-cf α 𝔄 𝔊 ⇌_CF exp-cat-cf α 𝔄 𝔉 :
      cat-FUNCT α ℭ 𝔄 ⇌⇌_Cβ cat-FUNCT α 𝔇 𝔄
proof–

  interpret β: 𝒵 β by (rule assms(1))
  interpret 𝔄: category α 𝔄 by (rule assms(3))

  show ?thesis
  proof
    (
      rule counit-unit-is-cf-adjunction(1)[
        where ε = ‹exp-cat-ntcf α 𝔄 (ε_C Φ)›
        ]
    )
    from assms is-cf-adjunction-axioms show
      exp-cat-ntcf α 𝔄 (η_C Φ) :
        cf-id (cat-FUNCT α ℭ 𝔄) ↦_CF exp-cat-cf α 𝔄 𝔉 ∘_CF exp-cat-cf α 𝔄 𝔊 :
        cat-FUNCT α ℭ 𝔄 ↦↦_Cβ cat-FUNCT α ℭ 𝔄
      by
        (
          cs-concl
            cs-simp:
              exp-cat-cf-cat-cf-id[symmetric] exp-cat-cf-cf-comp[symmetric]
            cs-intro: cat-small-cs-intros cat-FUNCT-cs-intros adj-cs-intros
        )
    from assms is-cf-adjunction-axioms show
      exp-cat-ntcf α 𝔄 (ε_C Φ) :
        exp-cat-cf α 𝔄 𝔊 ∘_CF exp-cat-cf α 𝔄 𝔉 ↦_CF cf-id (cat-FUNCT α 𝔇 𝔄) :
        cat-FUNCT α 𝔇 𝔄 ↦↦_Cβ cat-FUNCT α 𝔇 𝔄
      by
        (
          cs-concl
            cs-simp:
              exp-cat-cf-cat-cf-id[symmetric] exp-cat-cf-cf-comp[symmetric]
            cs-intro: cat-small-cs-intros cat-FUNCT-cs-intros adj-cs-intros
        )
    note [symmetric, cat-cs-simps] =
      ntcf-id-exp-cat-cf
      exp-cat-ntcf-ntcf-vcomp
```

```
    exp-cat-ntcf-ntcf-cf-comp
    exp-cat-ntcf-cf-ntcf-comp
  from assms show
    exp-cat-cf α 𝔄 𝔉 ∘_{CF−NTCF} exp-cat-ntcf α 𝔄 (ε_C Φ) ⊙_{NTCF}
      (exp-cat-ntcf α 𝔄 (η_C Φ) ∘_{NTCF−CF} exp-cat-cf α 𝔄 𝔉) =
      ntcf-id (exp-cat-cf α 𝔄 𝔉)
    by
      (
        cs-concl cs-shallow
          cs-simp: adj-cs-simps cat-cs-simps
          cs-intro: adj-cs-intros cat-cs-intros
      )
  from assms show
    exp-cat-ntcf α 𝔄 (ε_C Φ) ∘_{NTCF−CF} exp-cat-cf α 𝔄 𝔊 ⊙_{NTCF}
      (exp-cat-cf α 𝔄 𝔊 ∘_{CF−NTCF} exp-cat-ntcf α 𝔄 (η_C Φ)) =
        ntcf-id (exp-cat-cf α 𝔄 𝔊)
    by
      (
        cs-concl cs-shallow
          cs-simp: adj-cs-simps cat-cs-simps
          cs-intro: adj-cs-intros cat-cs-intros
      )
qed
  (
    use assms in
      ‹
        cs-concl
          cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      ›
  )+

qed
```

13.14 Adjoints on limits

```
lemma cf-AdjRight-preserves-limits:
  — See Chapter V-5 in [9].
  assumes Φ : 𝔉 ⇌_{CF} 𝔊 : 𝔛 ⇌⇌_{Cα} 𝔄
  shows is-cf-continuous α 𝔊
proof(intro is-cf-continuousI)

  interpret Φ: is-cf-adjunction α 𝔛 𝔄 𝔉 𝔊 Φ by (rule assms(1))

  show 𝔊 : 𝔄 ↦↦_{Cα} 𝔛 by (rule Φ.RL.is-functor-axioms)

  fix 𝔗 𝔍 assume prems: 𝔗 : 𝔍 ↦↦_{Cα} 𝔄

  show cf-preserves-limits α 𝔊 𝔗
  proof(intro cf-preserves-limitsI, rule prems, rule Φ.RL.is-functor-axioms)

    fix τ a assume τ : a <_{CF.lim} 𝔗 : 𝔍 ↦↦_{Cα} 𝔄
    then interpret τ: is-cat-limit α 𝔍 𝔄 𝔗 a τ .

    show 𝔊 ∘_{CF−NTCF} τ : 𝔊(‖ObjMap‖)(‖a‖) <_{CF.lim} 𝔊 ∘_{CF} 𝔗 : 𝔍 ↦↦_{Cα} 𝔛
    proof(intro is-cat-limitI)

      show 𝔊 ∘_{CF−NTCF} τ : 𝔊(‖ObjMap‖)(‖a‖) <_{CF.cone} 𝔊 ∘_{CF} 𝔗 : 𝔍 ↦↦_{Cα} 𝔛
        by
```

(
intro cf-ntcf-comp-cf-cat-cone prems,
rule τ*.is-cat-cone-axioms*,
intro Φ*.RL.is-functor-axioms*
)

fix σ' b' **assume** $\sigma' : b' <_{CF.cone} \mathfrak{G} \circ_{CF} \mathfrak{T} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{X}$
then interpret σ': *is-cat-cone* α b' $\mathfrak{J}$ $\mathfrak{X}$ $\langle \mathfrak{G} \circ_{CF} \mathfrak{T} \rangle$ σ' **.**

have ε_C $\Phi \circ_{NTCF-CF} \mathfrak{T} : \mathfrak{F} \circ_{CF} (\mathfrak{G} \circ_{CF} \mathfrak{T}) \mapsto_{CF} \mathfrak{T} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros adj-cs-intros*)
moreover have $\mathfrak{F} \circ_{CF-NTCF} \sigma' :$
$\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) <_{CF.cone} \mathfrak{F} \circ_{CF} (\mathfrak{G} \circ_{CF} \mathfrak{T}) : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by
(
intro cf-ntcf-comp-cf-cat-cone,
rule σ'*.is-cat-cone-axioms*,
rule Φ*.LR.is-functor-axioms*
)
ultimately have $(\varepsilon_C\ \Phi \circ_{NTCF-CF} \mathfrak{T}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma') :$
$\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) <_{CF.cone} \mathfrak{T} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by (*rule ntcf-vcomp-is-cat-cone*)
from τ*.cat-lim-unique-cone'*[*OF this*] **obtain** h
where h: $h : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) \mapsto_{\mathfrak{A}} a$
and $\varepsilon\mathfrak{T}$-$\mathfrak{F}\sigma'$: $\bigwedge j.\ j \in_\circ \mathfrak{J}(\lfloor Obj \rfloor) \Longrightarrow$
$((\varepsilon_C\ \Phi \circ_{NTCF-CF} \mathfrak{T}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma'))(\lfloor NTMap \rfloor)(\lfloor j \rfloor) = \tau(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} h$
and *h-unique*:
$[\![$
$h' : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) \mapsto_{\mathfrak{A}} a;$
$\bigwedge j.\ j \in_\circ \mathfrak{J}(\lfloor Obj \rfloor) \Longrightarrow$
$((\varepsilon_C\ \Phi \circ_{NTCF-CF} \mathfrak{T}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma'))(\lfloor NTMap \rfloor)(\lfloor j \rfloor) =$
$\tau(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} h'$
$]\!] \Longrightarrow h' = h$
for h'
by *metis*
have $\varepsilon\mathfrak{T}$-$\mathfrak{F}\sigma$:
$\varepsilon_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor \mathfrak{T}(\lfloor ObjMap \rfloor)(\lfloor j \rfloor) \rfloor) \circ_{A\mathfrak{A}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor \sigma'(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \rfloor) =$
$\tau(\lfloor NTMap \rfloor)(\lfloor j \rfloor) \circ_{A\mathfrak{A}} h$
if $j \in_\circ \mathfrak{J}(\lfloor Obj \rfloor)$ **for** j
using $\varepsilon\mathfrak{T}$-$\mathfrak{F}\sigma'$[*OF that*] *that*
by
(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros cat-cs-intros*
)

show $\exists! f'.$
$f' : b' \mapsto_{\mathfrak{X}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \wedge \sigma' = \mathfrak{G} \circ_{CF-NTCF} \tau \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{X}\ f'$
proof(*intro ex1I conjI*; (*elim conjE*) *?*)
let *?h'* = $\langle \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor h \rfloor) \circ_{A\mathfrak{X}} \eta_C\ \Phi(\lfloor NTMap \rfloor)(\lfloor b' \rfloor) \rangle$
from h **show** *?h'* : $b' \mapsto_{\mathfrak{X}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
by
(
cs-concl **cs-shallow**
cs-intro: *cat-cs-intros cat-lim-cs-intros adj-cs-intros*
)
show $\sigma' = \mathfrak{G} \circ_{CF-NTCF} \tau \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{X}$ *?h'*
proof(*rule ntcf-eqI*)

```
  show σ′ : cf-const 𝔍 𝔛 b′ ↦CF 𝔊 ∘CF 𝔗 : 𝔍 ↦↦Cα 𝔛
    by (rule σ′.is-ntcf-axioms)
  then have dom-lhs: 𝒟∘ (σ′(|NTMap|)) = 𝔍(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from h show
    𝔊 ∘CF−NTCF 𝔗 ⋆NTCF ntcf-const 𝔍 𝔛 ?h′ :
      cf-const 𝔍 𝔛 b′ ↦CF 𝔊 ∘CF 𝔗 : 𝔍 ↦↦Cα 𝔛
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-lim-cs-intros adj-cs-intros cat-cs-intros
      )
  then have dom-rhs:
    𝒟∘ ((𝔊 ∘CF−NTCF 𝔗 ⋆NTCF ntcf-const 𝔍 𝔛 ?h′)(|NTMap|)) = 𝔍(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps)
  show σ′(|NTMap|) = (𝔊 ∘CF−NTCF 𝔗 ⋆NTCF ntcf-const 𝔍 𝔛 ?h′)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix j assume prems′: j ∈∘ 𝔍(|Obj|)
    note [cat-cs-simps] = Φ.L.cat-assoc-helper
      [
          where h=‹𝔊(|ArrMap|)(|𝔗(|NTMap|)(|j|)|)›
            and g=‹𝔊(|ArrMap|)(|h|)›
            and f=‹ηC Φ(|NTMap|)(|b′|)›
            and q=‹𝔊(|ArrMap|)(|𝔗(|NTMap|)(|j|) ∘A𝔄 h|)›
      ]
    from prems′ h have [cat-cs-simps]:
      𝔊(|ArrMap|)(|𝔗(|NTMap|)(|j|) ∘A𝔄 h|) ∘A𝔛 ηC Φ(|NTMap|)(|b′|) = σ′(|NTMap|)(|j|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps εT-Fσ[OF prems′, symmetric]
            cs-intro: adj-cs-intros cat-cs-intros
        )
    from prems′ h show
      σ′(|NTMap|)(|j|) = (𝔊 ∘CF−NTCF 𝔗 ⋆NTCF ntcf-const 𝔍 𝔛 ?h′)(|NTMap|)(|j|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-lim-cs-intros adj-cs-intros cat-cs-intros
        )
  qed (cs-concl cs-intro: V-cs-intros cat-cs-intros)+
qed simp-all

fix f′ assume prems′:
  f′ : b′ ↦𝔛 𝔊(|ObjMap|)(|a|)
  σ′ = 𝔊 ∘CF−NTCF 𝔗 ⋆NTCF ntcf-const 𝔍 𝔛 f′

from prems′(2) have σ′-j-def′:
  σ′(|NTMap|)(|j|) = (𝔊 ∘CF−NTCF 𝔗 ⋆NTCF ntcf-const 𝔍 𝔛 f′)(|NTMap|)(|j|)
  for j
  by simp
have σ′-j-def: σ′(|NTMap|)(|j|) = 𝔊(|ArrMap|)(|𝔗(|NTMap|)(|j|)|) ∘A𝔛 f′
  if j ∈∘ 𝔍(|Obj|) for j
  using σ′-j-def′[of j] that prems′(1)
  by
    (
```

```
        cs-prems
          cs-simp: cat-cs-simps cs-intro: cat-lim-cs-intros cat-cs-intros
      )

    from prems'(1) have εa-𝔉f':
      ε_C Φ(|NTMap|)(|a|) ∘_A𝔄 𝔉(|ArrMap|)(|f'|) : 𝔉(|ObjMap|)(|b'|) ↦_𝔄 a
      by (cs-concl cs-intro: cat-lim-cs-intros cat-cs-intros adj-cs-intros)

    interpret ε: is-ntcf α 𝔄 𝔄 ‹𝔉 ∘_CF 𝔊› ‹cf-id 𝔄› ‹ε_C Φ›
      by (rule Φ.cf-adjunction-counit-is-ntcf)

    have
      (ε_C Φ ∘_NTCF-CF 𝔗 ·_NTCF (𝔉 ∘_CF-NTCF σ'))(|NTMap|)(|j|) =
        τ(|NTMap|)(|j|) ∘_A𝔄 (ε_C Φ(|NTMap|)(|a|) ∘_A𝔄 𝔉(|ArrMap|)(|f'|))
      if j ∈_∘ 𝔍(|Obj|) for j
    proof–
      from that have τ(|NTMap|)(|j|) : a ↦_𝔄 𝔗(|ObjMap|)(|j|)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros
          )
      from ε.ntcf-Comp-commute[OF this] that have [cat-cs-simps]:
        ε_C Φ(|NTMap|)(|𝔗(|ObjMap|)(|j|)|) ∘_A𝔄 𝔉(|ArrMap|)(|𝔊(|ArrMap|)(|τ(|NTMap|)(|j|)|)|) =
          τ(|NTMap|)(|j|) ∘_A𝔄 ε_C Φ(|NTMap|)(|a|)
        by
          (
            cs-prems cs-shallow
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros
          )
      note [cat-cs-simps] = Φ.R.cat-assoc-helper
        [
          where h=‹ε_C Φ(|NTMap|)(|𝔗(|ObjMap|)(|j|)|)›
            and g=‹𝔉(|ArrMap|)(|𝔊(|ArrMap|)(|τ(|NTMap|)(|j|)|)|)›
            and q=‹τ(|NTMap|)(|j|) ∘_A𝔄 ε_C Φ(|NTMap|)(|a|)›
        ]
      show ?thesis
      using that prems'(1)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps σ'-j-def
              cs-intro: cat-lim-cs-intros cat-cs-intros adj-cs-intros
          )
      qed
      from h-unique[OF εa-𝔉f' this] have
        𝔊(|ArrMap|)(|ε_C Φ(|NTMap|)(|a|) ∘_A𝔄 𝔉(|ArrMap|)(|f'|)|) ∘_A𝔛 η_C Φ(|NTMap|)(|b'|) = ?h'
        by simp
      from this prems'(1) show f' = 𝔊(|ArrMap|)(|h|) ∘_A𝔛 η_C Φ(|NTMap|)(|b'|)
        by
          (
            cs-prems
              cs-simp: cat-cs-simps Φ.cf-adj-counit-unit-app
              cs-intro: cat-lim-cs-intros cat-cs-intros
          )
    qed

  qed
```

qed

qed

14 Simple Kan extensions

14.1 Background

named-theorems *cat-Kan-cs-simps*
named-theorems *cat-Kan-cs-intros*

14.2 Kan extension

14.2.1 Definition and elementary properties

See Chapter X-3 in [9].

locale *is-cat-rKe* =
  AG: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ +
  Ran: *is-functor* α $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{G}$ +
  ntcf-rKe: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{A}$ $\langle \mathfrak{G} \circ_{CF} \mathfrak{K} \rangle$ $\mathfrak{T}$ ε
  for α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ ε +
  assumes *cat-rKe-ua-fo*:
    universal-arrow-fo
      (*exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{K}$)
      (*cf-map* $\mathfrak{T}$)
      (*cf-map* $\mathfrak{G}$)
      (*ntcf-arrow* ε)

syntax *-is-cat-rKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  $(\langle (- :/ - \circ_{CF} - \mapsto_{CF.rKe^{\mathfrak{l}}} - :/ - \mapsto_{C} - \mapsto_{C} -) \rangle$ [*51*, *51*, *51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-rKe* $\rightleftharpoons$ *is-cat-rKe*
translations $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} \mathfrak{A} \rightleftharpoons$
  CONST is-cat-rKe α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ ε

locale *is-cat-lKe* =
  AG: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ +
  Lan: *is-functor* α $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{F}$ +
  ntcf-lKe: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ $\langle \mathfrak{F} \circ_{CF} \mathfrak{K} \rangle$ η
  for α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ η +
  assumes *cat-lKe-ua-fo*:
    universal-arrow-fo
      (*exp-cat-cf* α (*op-cat* $\mathfrak{A}$) (*op-cf* $\mathfrak{K}$))
      (*cf-map* $\mathfrak{T}$)
      (*cf-map* $\mathfrak{F}$)
      (*ntcf-arrow* (*op-ntcf* η))

syntax *-is-cat-lKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  $(\langle (- :/ - \mapsto_{CF.lKe^{\mathfrak{l}}} - \circ_{CF} - :/ - \mapsto_{C} - \mapsto_{C} -) \rangle$ [*51*, *51*, *51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-lKe* $\rightleftharpoons$ *is-cat-lKe*
translations $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} \mathfrak{A} \rightleftharpoons$
  CONST is-cat-lKe α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ η

Rules.

lemma (**in** *is-cat-rKe*) *is-cat-rKe-axioms'*[*cat-Kan-cs-intros*]:
  assumes $\alpha' = \alpha$
    and $\mathfrak{G}' = \mathfrak{G}$
    and $\mathfrak{K}' = \mathfrak{K}$
    and $\mathfrak{T}' = \mathfrak{T}$
    and $\mathfrak{B}' = \mathfrak{B}$
    and $\mathfrak{A}' = \mathfrak{A}$
    and $\mathfrak{C}' = \mathfrak{C}$
  shows $\varepsilon : \mathfrak{G}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe_{\alpha'}} \mathfrak{T}' : \mathfrak{B}' \mapsto_{C} \mathfrak{C}' \mapsto_{C} \mathfrak{A}'$

unfolding *assms* **by** (*rule is-cat-rKe-axioms*)

mk-ide rf *is-cat-rKe-def*[*unfolded is-cat-rKe-axioms-def*]
|*intro is-cat-rKeI*|
|*dest is-cat-rKeD*[*dest*]|
|*elim is-cat-rKeE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-rKeD*(*1–3*)

lemma (**in** *is-cat-lKe*) *is-cat-lKe-axioms'*[*cat-Kan-cs-intros*]:
assumes $\alpha' = \alpha$
and $\mathfrak{F}' = \mathfrak{F}$
and $\mathfrak{K}' = \mathfrak{K}$
and $\mathfrak{T}' = \mathfrak{T}$
and $\mathfrak{B}' = \mathfrak{B}$
and $\mathfrak{A}' = \mathfrak{A}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\eta : \mathfrak{T}' \mapsto_{CF.lKe\alpha} \mathfrak{F}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_{C} \mathfrak{C}' \mapsto_{C} \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-lKe-axioms*)

mk-ide rf *is-cat-lKe-def*[*unfolded is-cat-lKe-axioms-def*]
|*intro is-cat-lKeI*|
|*dest is-cat-lKeD*[*dest*]|
|*elim is-cat-lKeE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-lKeD*(*1–3*)

Duality.

lemma (**in** *is-cat-rKe*) *is-cat-lKe-op*:
op-ntcf ε :
op-cf $\mathfrak{T} \mapsto_{CF.lKe\alpha}$ *op-cf* $\mathfrak{G} \circ_{CF}$ *op-cf* $\mathfrak{K}$:
op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C}$ *op-cat* $\mathfrak{A}$
by (*intro is-cat-lKeI*, *unfold cat-op-simps*; (*intro cat-rKe-ua-fo*)*?*)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)+

lemma (**in** *is-cat-rKe*) *is-cat-lKe-op'*[*cat-op-intros*]:
assumes $\mathfrak{T}' =$ *op-cf* $\mathfrak{T}$
and $\mathfrak{G}' =$ *op-cf* $\mathfrak{G}$
and $\mathfrak{K}' =$ *op-cf* $\mathfrak{K}$
and $\mathfrak{B}' =$ *op-cat* $\mathfrak{B}$
and $\mathfrak{A}' =$ *op-cat* $\mathfrak{A}$
and $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$
shows *op-ntcf* $\varepsilon : \mathfrak{T}' \mapsto_{CF.lKe\alpha} \mathfrak{G}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_{C} \mathfrak{C}' \mapsto_{C} \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-lKe-op*)

lemmas [*cat-op-intros*] = *is-cat-rKe.is-cat-lKe-op'*

lemma (**in** *is-cat-lKe*) *is-cat-rKe-op*:
op-ntcf η :
op-cf $\mathfrak{F} \circ_{CF}$ *op-cf* $\mathfrak{K} \mapsto_{CF.rKe\alpha}$ *op-cf* $\mathfrak{T}$:
op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C}$ *op-cat* $\mathfrak{A}$
by (*intro is-cat-rKeI*, *unfold cat-op-simps*; (*intro cat-lKe-ua-fo*)*?*)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)+

lemma (**in** *is-cat-lKe*) *is-cat-lKe-op'*[*cat-op-intros*]:
assumes $\mathfrak{T}' =$ *op-cf* $\mathfrak{T}$
and $\mathfrak{F}' =$ *op-cf* $\mathfrak{F}$
and $\mathfrak{K}' =$ *op-cf* $\mathfrak{K}$

and $\mathfrak{B}' = op\text{-}cat\ \mathfrak{B}$
and $\mathfrak{A}' = op\text{-}cat\ \mathfrak{A}$
and $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows $op\text{-}ntcf\ \eta : \mathfrak{F}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe\alpha} \mathfrak{T}' : \mathfrak{B}' \mapsto_{C} \mathfrak{C}' \mapsto_{C} \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-rKe-op*)

lemmas [*cat-op-intros*] = *is-cat-lKe.is-cat-lKe-op'*

Elementary properties.

lemma (**in** *is-cat-rKe*) *cat-rKe-exp-cat-cf-cat-FUNCT-is-arr*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows $exp\text{-}cat\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{K} : cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A} \mapsto\mapsto_{C.tiny\beta} cat\text{-}FUNCT\ \alpha\ \mathfrak{B}\ \mathfrak{A}$
by
(
rule exp-cat-cf-is-tiny-functor[
OF assms Ran.HomCod.category-axioms AG.is-functor-axioms
]
)

lemma (**in** *is-cat-lKe*) *cat-lKe-exp-cat-cf-cat-FUNCT-is-arr*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows $exp\text{-}cat\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{K} : cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A} \mapsto\mapsto_{C.tiny\beta} cat\text{-}FUNCT\ \alpha\ \mathfrak{B}\ \mathfrak{A}$
by
(
rule exp-cat-cf-is-tiny-functor[
OF assms Lan.HomCod.category-axioms AG.is-functor-axioms
]
)

14.2.2 Universal property

See Chapter X-3 in [9] and [2][8].

lemma *is-cat-rKeI'*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $\bigwedge \mathfrak{G}'\ \varepsilon'.$
$[\![\ \mathfrak{G}' : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A};\ \varepsilon' : \mathfrak{G}' \circ_{CF} \mathfrak{K} \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}\ ]\!] \Longrightarrow$
$\exists!\sigma.\ \sigma : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A} \wedge \varepsilon' = \varepsilon \bullet_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K})$
shows $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} \mathfrak{A}$
proof–
interpret $\mathfrak{K}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{A}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
interpret ε: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \langle \mathfrak{G} \circ_{CF} \mathfrak{K} \rangle\ \mathfrak{T}\ \varepsilon$ **by** (*rule assms*(*3*))
let $?\mathfrak{A}\mathfrak{K} = \langle exp\text{-}cat\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{K} \rangle$
and $?\mathfrak{T} = \langle cf\text{-}map\ \mathfrak{T} \rangle$
and $?\mathfrak{G} = \langle cf\text{-}map\ \mathfrak{G} \rangle$
show *?thesis*
proof(*intro is-cat-rKeI is-functor.universal-arrow-foI assms*)
define β **where** $\beta = \alpha + \omega$
have $\mathcal{Z}\ \beta$ **and** $\alpha\beta$: $\alpha \in_{\circ} \beta$
by (*simp-all add*: β*-def* $\mathfrak{K}.\mathcal{Z}$*-Limit-*$\alpha\omega$ $\mathfrak{K}.\mathcal{Z}$*-*ω*-*$\alpha\omega$ $\mathcal{Z}$*-def* $\mathfrak{K}.\mathcal{Z}$*-*α*-*$\alpha\omega$)
then interpret β: $\mathcal{Z}\ \beta$ **by** *simp*
show $?\mathfrak{A}\mathfrak{K} : cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A} \mapsto\mapsto_{C\beta} cat\text{-}FUNCT\ \alpha\ \mathfrak{B}\ \mathfrak{A}$
by
(

[8] https://en.wikipedia.org/wiki/Kan_extension

```
    cs-concl cs-shallow cs-intro:
      cat-small-cs-intros
      exp-cat-cf-is-tiny-functor[
        OF β.Z-axioms αβ 𝔊.HomCod.category-axioms assms(1)
        ]
  )
from αβ assms(2) show cf-map 𝔊 ∈∘ cat-FUNCT α 𝔈 𝔄(|Obj|)
  unfolding cat-FUNCT-components
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-FUNCT-cs-intros)
from assms(1–3) show ntcf-arrow ε :
  ?𝔄𝔎(|ObjMap|)(|?𝔊|) ↦cat-FUNCT α 𝔅 𝔄 ?𝔗
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-Kan-cs-simps cat-FUNCT-cs-simps cat-FUNCT-components(1)
        cs-intro: cat-FUNCT-cs-intros
    )
fix 𝔉′ ε′ assume prems:
  𝔉′ ∈∘ cat-FUNCT α 𝔈 𝔄(|Obj|)
  ε′ : ?𝔄𝔎(|ObjMap|)(|𝔉′|) ↦cat-FUNCT α 𝔅 𝔄 ?𝔗
from prems(1) have 𝔉′ ∈∘ cf-maps α 𝔈 𝔄
  unfolding cat-FUNCT-components(1) by simp
then obtain 𝔉 where 𝔉′-def: 𝔉′ = cf-map 𝔉 and 𝔉: 𝔉 : 𝔈 ↦↦Cα 𝔄
  by clarsimp
note ε′ = cat-FUNCT-is-arrD[OF prems(2)]
from ε′(1) 𝔉 have ε′-is-ntcf:
  ntcf-of-ntcf-arrow 𝔅 𝔄 ε′ : 𝔉 ∘CF 𝔎 ↦CF 𝔗 : 𝔅 ↦↦Cα 𝔄
  by
    (
      cs-prems
        cs-simp: 𝔉′-def cat-Kan-cs-simps cat-FUNCT-cs-simps
        cs-intro: cat-cs-intros cat-FUNCT-cs-intros
    )
from assms(4)[OF 𝔉 ε′-is-ntcf] obtain σ
  where σ: σ : 𝔉 ↦CF 𝔊 : 𝔈 ↦↦Cα 𝔄
    and ε′-def′: ntcf-of-ntcf-arrow 𝔅 𝔄 ε′ = ε ⋆NTCF (σ ∘NTCF−CF 𝔎)
    and unique-σ: ⋀σ′.
      ⟦
        σ′ : 𝔉 ↦CF 𝔊 : 𝔈 ↦↦Cα 𝔄;
        ntcf-of-ntcf-arrow 𝔅 𝔄 ε′ = ε ⋆NTCF (σ′ ∘NTCF−CF 𝔎)
      ⟧ ⟹ σ′ = σ
  by metis
show ∃!f′.
  f′ : 𝔉′ ↦cat-FUNCT α 𝔈 𝔄 ?𝔊 ∧
  ε′ = umap-fo ?𝔄𝔎 ?𝔗 ?𝔊 (ntcf-arrow ε) 𝔉′(|ArrVal|)(|f′|)
proof(intro ex1I conjI; (elim conjE)?, unfold 𝔉′-def)
  from σ show ntcf-arrow σ : cf-map 𝔉 ↦cat-FUNCT α 𝔈 𝔄 ?𝔊
    by (cs-concl cs-shallow cs-intro: cat-FUNCT-cs-intros)
  from αβ assms(1–3) σ ε′(1) show
    ε′ = umap-fo ?𝔄𝔎 ?𝔗 ?𝔊 (ntcf-arrow ε) (cf-map 𝔉)(|ArrVal|)(|ntcf-arrow σ|)
    by (subst ε′)
      (
        cs-concl
          cs-simp:
            ε′-def′[symmetric]
            cat-cs-simps
            cat-FUNCT-cs-simps
            cat-Kan-cs-simps
```

```
            cs-intro:
              cat-small-cs-intros
              cat-cs-intros
              cat-Kan-cs-intros
              cat-FUNCT-cs-intros
          )
      fix σ′ assume prems:
        σ′ : cf-map 𝔉 ↦_cat-FUNCT α 𝔈 𝔄 ?𝔊
        ε′ = umap-fo ?𝔄𝔎 ?𝔗 ?𝔊 (ntcf-arrow ε) (cf-map 𝔉)(⦇ArrVal⦈)(⦇σ′⦈)
      note σ′ = cat-FUNCT-is-arrD[OF prems(1)]
      from σ′(1) 𝔉 have ntcf-of-ntcf-arrow 𝔈 𝔄 σ′ : 𝔉 ↦_CF 𝔊 : 𝔈 ↦↦_Cα 𝔄
        by
          (
            cs-prems cs-shallow
              cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
          )
      moreover from prems(2) prems(1) αβ assms(1–3) this ε′(1) have
        ntcf-of-ntcf-arrow 𝔅 𝔄 ε′ =
          ε ⋆_NTCF (ntcf-of-ntcf-arrow 𝔈 𝔄 σ′ ∘_NTCF-CF 𝔎)
        by (subst (asm) ε′(2))
          (
            cs-prems
              cs-simp: cat-Kan-cs-simps cat-FUNCT-cs-simps cat-cs-simps
              cs-intro:
                cat-Kan-cs-intros
                cat-small-cs-intros
                cat-cs-intros
                cat-FUNCT-cs-intros
          )
      ultimately have σ-def: σ = ntcf-of-ntcf-arrow 𝔈 𝔄 σ′
        by (rule unique-σ[symmetric])
      show σ′ = ntcf-arrow σ
        by (subst σ′(2), use nothing in ‹subst σ-def›)
          (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed
  qed
qed

lemma is-cat-lKeI′:
  assumes 𝔎 : 𝔅 ↦↦_Cα 𝔈
    and 𝔉 : 𝔈 ↦↦_Cα 𝔄
    and η : 𝔗 ↦_CF 𝔉 ∘_CF 𝔎 : 𝔅 ↦↦_Cα 𝔄
    and ⋀𝔉′ η′.
      ⟦ 𝔉′ : 𝔈 ↦↦_Cα 𝔄; η′ : 𝔗 ↦_CF 𝔉′ ∘_CF 𝔎 : 𝔅 ↦↦_Cα 𝔄 ⟧ ⟹
        ∃!σ. σ : 𝔉 ↦_CF 𝔉′ : 𝔈 ↦↦_Cα 𝔄 ∧ η′ = (σ ∘_NTCF-CF 𝔎) ⋆_NTCF η
  shows η : 𝔗 ↦_CF.lKeα 𝔉 ∘_CF 𝔎 : 𝔅 ↦_C 𝔈 ↦_C 𝔄
proof–
  interpret 𝔎: is-functor α 𝔅 𝔈 𝔎 by (rule assms(1))
  interpret 𝔉: is-functor α 𝔈 𝔄 𝔉 by (rule assms(2))
  interpret η: is-ntcf α 𝔅 𝔄 𝔗 ‹𝔉 ∘_CF 𝔎› η by (rule assms(3))
  have
    ∃!σ.
      σ : 𝔊′ ↦_CF op-cf 𝔉 : op-cat 𝔈 ↦↦_Cα op-cat 𝔄 ∧
      η′ = op-ntcf η ⋆_NTCF (σ ∘_NTCF-CF op-cf 𝔎)
    if 𝔊′ : op-cat 𝔈 ↦↦_Cα op-cat 𝔄
      and η′ : 𝔊′ ∘_CF op-cf 𝔎 ↦_CF op-cf 𝔗 : op-cat 𝔅 ↦↦_Cα op-cat 𝔄
    for 𝔊′ η′
  proof–
```

```
  interpret 𝔊′: is-functor α ‹op-cat ℭ› ‹op-cat 𝔄› 𝔊′ by (rule that(1))
  interpret η′:
    is-ntcf α ‹op-cat 𝔅› ‹op-cat 𝔄› ‹𝔊′ ∘_CF op-cf 𝔎› ‹op-cf 𝔗› η′
    by (rule that(2))
  from assms(4)[
      OF is-functor.is-functor-op[OF that(1), unfolded cat-op-simps],
      OF is-ntcf.is-ntcf-op[OF that(2), unfolded cat-op-simps]
      ]
  obtain σ where σ: σ : 𝔉 ↦_CF op-cf 𝔊′ : ℭ ↦↦_Cα 𝔄
    and op-η′-def: op-ntcf η′ = σ ∘_NTCF-CF 𝔎 ⋆_NTCF η
    and unique-σ′:
      ⟦
        σ′ : 𝔉 ↦_CF op-cf 𝔊′ : ℭ ↦↦_Cα 𝔄;
        op-ntcf η′ = σ′ ∘_NTCF-CF 𝔎 ⋆_NTCF η
      ⟧ ⟹ σ′ = σ
    for σ′
    by metis
  interpret σ: is-ntcf α ℭ 𝔄 𝔉 ‹op-cf 𝔊′› σ by (rule σ)
  show ?thesis
  proof(intro ex1I conjI; (elim conjE)?)
    show op-ntcf σ : 𝔊′ ↦_CF op-cf 𝔉 : op-cat ℭ ↦↦_Cα op-cat 𝔄
      by (rule σ.is-ntcf-op[unfolded cat-op-simps])
    from op-η′-def have op-ntcf (op-ntcf η′) = op-ntcf (σ ∘_NTCF-CF 𝔎 ⋆_NTCF η)
      by simp
    from this σ assms(1–3) show η′-def:
      η′ = op-ntcf η ⋆_NTCF (op-ntcf σ ∘_NTCF-CF op-cf 𝔎)
      by (cs-prems cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros)
    fix σ′ assume prems:
      σ′ : 𝔊′ ↦_CF op-cf 𝔉 : op-cat ℭ ↦↦_Cα op-cat 𝔄
      η′ = op-ntcf η ⋆_NTCF (σ′ ∘_NTCF-CF op-cf 𝔎)
    interpret σ′: is-ntcf α ‹op-cat ℭ› ‹op-cat 𝔄› 𝔊′ ‹op-cf 𝔉› σ′
      by (rule prems(1))
    from prems(2) have
      op-ntcf η′ = op-ntcf (op-ntcf η ⋆_NTCF (σ′ ∘_NTCF-CF op-cf 𝔎))
      by simp
    also have ... = op-ntcf σ′ ∘_NTCF-CF 𝔎 ⋆_NTCF η
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
        )
    finally have op-ntcf η′ = op-ntcf σ′ ∘_NTCF-CF 𝔎 ⋆_NTCF η by simp
    from unique-σ′[OF σ′.is-ntcf-op[unfolded cat-op-simps] this] show
      σ′ = op-ntcf σ
      by (auto simp: cat-op-simps)
  qed
qed
from
  is-cat-rKeI′
    [
      OF 𝔎.is-functor-op 𝔉.is-functor-op η.is-ntcf-op[unfolded cat-op-simps],
      unfolded cat-op-simps,
      OF this
    ]
interpret η: is-cat-rKe
  α
  ‹op-cat 𝔅›
```

```
    ‹op-cat 𝔈›
    ‹op-cat 𝔄›
    ‹op-cf 𝔎›
    ‹op-cf 𝔗›
    ‹op-cf 𝔉›
    ‹op-ntcf η›
    by simp
  show η : 𝔗 ↦_{CF.lKeα} 𝔉 ∘_{CF} 𝔎 : 𝔅 ↦_C 𝔈 ↦_C 𝔄
    by (rule η.is-cat-lKe-op[unfolded cat-op-simps])
qed

lemma (in is-cat-rKe) cat-rKe-unique:
  assumes 𝔊' : 𝔈 ↦↦_{Cα} 𝔄 and ε' : 𝔊' ∘_{CF} 𝔎 ↦_{CF} 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
  shows ∃!σ. σ : 𝔊' ↦_{CF} 𝔊 : 𝔈 ↦↦_{Cα} 𝔄 ∧ ε' = ε ∙_{NTCF} (σ ∘_{NTCF-CF} 𝔎)
proof–

  interpret 𝔊': is-functor α 𝔈 𝔄 𝔊' by (rule assms(1))
  interpret ε': is-ntcf α 𝔅 𝔄 ‹𝔊' ∘_{CF} 𝔎› 𝔗 ε' by (rule assms(2))

  let ?𝔗 = ‹cf-map 𝔗›
    and ?𝔊 = ‹cf-map 𝔊›
    and ?𝔊' = ‹cf-map 𝔊'›
    and ?ε = ‹ntcf-arrow ε›
    and ?ε' = ‹ntcf-arrow ε'›

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def AG.𝒵-Limit-αω AG.𝒵-ω-αω 𝒵-def AG.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  interpret 𝔄𝔎: is-tiny-functor
    β ‹cat-FUNCT α 𝔈 𝔄› ‹cat-FUNCT α 𝔅 𝔄› ‹exp-cat-cf α 𝔄 𝔎›
    by (rule cat-rKe-exp-cat-cf-cat-FUNCT-is-arr[OF β.𝒵-axioms αβ])

  from assms(1) have 𝔊': ?𝔊' ∈∘ cat-FUNCT α 𝔈 𝔄(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-components(1) cs-intro: cat-FUNCT-cs-intros
      )
  with assms(2) have
    ?ε' : exp-cat-cf α 𝔄 𝔎(|ObjMap|)(|?𝔊'|) ↦_{cat-FUNCT α 𝔅 𝔄} ?𝔗
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )

  from
    is-functor.universal-arrow-foD(3)[
      OF 𝔄𝔎.is-functor-axioms cat-rKe-ua-fo 𝔊' this
      ]
  obtain f' where f': f' : cf-map 𝔊' ↦_{cat-FUNCT α 𝔈 𝔄} cf-map 𝔊
    and ε'-def: ?ε' = umap-fo (exp-cat-cf α 𝔄 𝔎) ?𝔗 ?𝔊 ?ε ?𝔊'(|ArrVal|)(|f'|)
    and f'-unique:
      ⟦
        f'' : ?𝔊' ↦_{cat-FUNCT α 𝔈 𝔄} ?𝔊;
```

```
      ntcf-arrow ε′ = umap-fo (exp-cat-cf α 𝔄 𝔎) ?𝔗 ?𝔊 ?ε ?𝔊′(⦇ArrVal⦈)(⦇f′′⦈)
    ⟧ ⟹ f′′ = f′
  for f′′
  by metis

 show ?thesis
 proof(intro ex1I conjI; (elim conjE)?)
  from ε′-def cat-FUNCT-is-arrD(1)[OF f′] show
   ε′ = ε ∙NTCF (ntcf-of-ntcf-arrow ℭ 𝔄 f′ ∘NTCF−CF 𝔎)
   by (subst (asm) cat-FUNCT-is-arrD(2)[OF f′])
    (
     cs-prems cs-shallow
      cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-Kan-cs-simps
      cs-intro: cat-cs-intros cat-FUNCT-cs-intros
    )
  from cat-FUNCT-is-arrD(1)[OF f′] show f′-is-arr:
   ntcf-of-ntcf-arrow ℭ 𝔄 f′ : 𝔊′ ↦CF 𝔊 : ℭ ↦↦Cα 𝔄
   by
    (
     cs-prems cs-shallow
      cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
    )
  fix σ assume prems:
   σ : 𝔊′ ↦CF 𝔊 : ℭ ↦↦Cα 𝔄 ε′ = ε ∙NTCF (σ ∘NTCF−CF 𝔎)
  interpret σ: is-ntcf α ℭ 𝔄 𝔊′ 𝔊 σ by (rule prems(1))
  from prems(1) have σ:
   ntcf-arrow σ : cf-map 𝔊′ ↦cat-FUNCT α ℭ 𝔄 cf-map 𝔊
   by (cs-concl cs-shallow cs-intro: cat-FUNCT-cs-intros)
  from prems have ε′-def: ntcf-arrow ε′ =
   umap-fo (exp-cat-cf α 𝔄 𝔎) ?𝔗 ?𝔊 ?ε ?𝔊′(⦇ArrVal⦈)(⦇ntcf-arrow σ⦈)
   by
    (
     cs-concl cs-shallow
      cs-simp: prems(2) cat-Kan-cs-simps cat-cs-simps cat-FUNCT-cs-simps
      cs-intro: cat-cs-intros cat-FUNCT-cs-intros
    )
  show σ = ntcf-of-ntcf-arrow ℭ 𝔄 f′
   unfolding f′-unique[OF σ ε′-def, symmetric]
   by
    (
     cs-concl cs-shallow
      cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
    )
 qed

qed

lemma (in is-cat-lKe) cat-lKe-unique:
 assumes 𝔉′ : ℭ ↦↦Cα 𝔄 and η′ : 𝔗 ↦CF 𝔉′ ∘CF 𝔎 : 𝔅 ↦↦Cα 𝔄
 shows ∃!σ. σ : 𝔉 ↦CF 𝔉′ : ℭ ↦↦Cα 𝔄 ∧ η′ = (σ ∘NTCF−CF 𝔎) ∙NTCF η
proof–

 interpret 𝔉′: is-functor α ℭ 𝔄 𝔉′ by (rule assms(1))
 interpret η′: is-ntcf α 𝔅 𝔄 𝔗 ‹𝔉′ ∘CF 𝔎› η′ by (rule assms(2))
 interpret η: is-cat-rKe
  α ‹op-cat 𝔅› ‹op-cat ℭ› ‹op-cat 𝔄› ‹op-cf 𝔎› ‹op-cf 𝔗› ‹op-cf 𝔉› ‹op-ntcf η›
  by (rule is-cat-rKe-op)
```

```
  from η.cat-rKe-unique[OF 𝔉'.is-functor-op η'.is-ntcf-op[unfolded cat-op-simps]]
  obtain σ where σ: σ : op-cf 𝔉' ↦_CF op-cf 𝔉 : op-cat ℭ ↦↦_Cα op-cat 𝔄
    and η'-def: op-ntcf η' = op-ntcf η ⋆_NTCF (σ ∘_NTCF-CF op-cf 𝔎)
    and unique-σ': ⋀σ'.
      ⟦
        σ' : op-cf 𝔉' ↦_CF op-cf 𝔉 : op-cat ℭ ↦↦_Cα op-cat 𝔄;
        op-ntcf η' = op-ntcf η ⋆_NTCF (σ' ∘_NTCF-CF op-cf 𝔎)
      ⟧ ⟹ σ' = σ
    by metis

  interpret σ: is-ntcf α ‹op-cat ℭ› ‹op-cat 𝔄› ‹op-cf 𝔉'› ‹op-cf 𝔉› σ
    by (rule σ)

  show ?thesis
  proof(intro ex1I conjI; (elim conjE)?)
    show op-ntcf σ : 𝔉 ↦_CF 𝔉' : ℭ ↦↦_Cα 𝔄
      by (rule σ.is-ntcf-op[unfolded cat-op-simps])
    have η' = op-ntcf (op-ntcf η')
      by (cs-concl cs-shallow cs-simp: cat-op-simps)
    also from η'-def have ... = op-ntcf (op-ntcf η ⋆_NTCF (σ ∘_NTCF-CF op-cf 𝔎))
      by simp
    also have ... = op-ntcf σ ∘_NTCF-CF 𝔎 ⋆_NTCF η
      by (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros)
    finally show η' = op-ntcf σ ∘_NTCF-CF 𝔎 ⋆_NTCF η by simp
    fix σ' assume prems:
      σ' : 𝔉 ↦_CF 𝔉' : ℭ ↦↦_Cα 𝔄
      η' = σ' ∘_NTCF-CF 𝔎 ⋆_NTCF η
    interpret σ': is-ntcf α ℭ 𝔄 𝔉 𝔉' σ' by (rule prems(1))
    from prems(2) have op-ntcf η' = op-ntcf (σ' ∘_NTCF-CF 𝔎 ⋆_NTCF η)
      by simp
    also have ... = op-ntcf η ⋆_NTCF (op-ntcf σ' ∘_NTCF-CF op-cf 𝔎)
      by (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros)
    finally have op-ntcf η' = op-ntcf η ⋆_NTCF (op-ntcf σ' ∘_NTCF-CF op-cf 𝔎)
      by simp
    from unique-σ'[OF σ'.is-ntcf-op this] show σ' = op-ntcf σ
      by (auto simp: cat-op-simps)
  qed

qed
```

14.2.3 Further properties

```
lemma (in is-cat-rKe) cat-rKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit:
  assumes 𝒵 β and α ∈₀ β
  shows
    ntcf-ua-fo β (exp-cat-cf α 𝔄 𝔎) (cf-map 𝔗) (cf-map 𝔊) (ntcf-arrow ε) :
      Hom_O.Cβ cat-FUNCT α ℭ 𝔄(–,cf-map 𝔊) ↦_CF.iso
      Hom_O.Cβ cat-FUNCT α 𝔅 𝔄(–,cf-map 𝔗) ∘_CF op-cf (exp-cat-cf α 𝔄 𝔎) :
      op-cat (cat-FUNCT α ℭ 𝔄) ↦↦_Cβ cat-Set β
proof–
  interpret 𝔄-𝔎:
    is-tiny-functor β ‹cat-FUNCT α ℭ 𝔄› ‹cat-FUNCT α 𝔅 𝔄› ‹exp-cat-cf α 𝔄 𝔎›
    by
      (
        rule exp-cat-cf-is-tiny-functor[
          OF assms Ran.HomCod.category-axioms AG.is-functor-axioms
          ]
      )
```

```
  show ?thesis
    by
      (
        rule is-functor.cf-ntcf-ua-fo-is-iso-ntcf[
          OF 𝔄-ℜ.is-functor-axioms cat-rKe-ua-fo
          ]
      )
qed

lemma (in is-cat-lKe) cat-lKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  defines 𝔄ℜ ≡ exp-cat-cf α (op-cat 𝔄) (op-cf ℜ)
    and 𝔄ℭ ≡ cat-FUNCT α (op-cat ℭ) (op-cat 𝔄)
    and 𝔄𝔅 ≡ cat-FUNCT α (op-cat 𝔅) (op-cat 𝔄)
  shows
    ntcf-ua-fo β 𝔄ℜ (cf-map 𝔗) (cf-map 𝔉) (ntcf-arrow (op-ntcf η)) :
      Hom_{O.Cβ}𝔄ℭ(−,cf-map 𝔉) ↦_{CF.iso} Hom_{O.Cβ}𝔄𝔅(−,cf-map 𝔗) ∘_{CF} op-cf 𝔄ℜ :
      op-cat 𝔄ℭ ↦↦_{Cβ} cat-Set β
proof–
  note simps = 𝔄ℭ-def 𝔄𝔅-def 𝔄ℜ-def
  interpret 𝔄-ℜ: is-tiny-functor β 𝔄ℭ 𝔄𝔅 𝔄ℜ
    unfolding simps
    by
      (
        rule exp-cat-cf-is-tiny-functor[
          OF assms(1,2) Lan.HomCod.category-op AG.is-functor-op
          ]
      )
  show ?thesis
    unfolding simps
    by
      (
        rule is-functor.cf-ntcf-ua-fo-is-iso-ntcf[
          OF 𝔄-ℜ.is-functor-axioms[unfolded simps] cat-lKe-ua-fo
          ]
      )
qed
```

14.3 Opposite universal arrow for Kan extensions

14.3.1 Definition and elementary properties

The following definition is merely a convenience utility for the exposition of dual results associated with the formula for the right Kan extension and the pointwise right Kan extension.

```
definition op-ua :: (V ⇒ V) ⇒ V ⇒ V ⇒ V
  where op-ua lim-Obj ℜ c =
    [
      lim-Obj c(|UObj|),
      op-ntcf (lim-Obj c(|UArr|)) ∘_{NTCF−CF} inv-cf (op-cf-obj-comma ℜ c)
    ]∘
```

Components.

```
lemma op-ua-components:
  shows [cat-op-simps]: op-ua lim-Obj ℜ c(|UObj|) = lim-Obj c(|UObj|)
    and op-ua lim-Obj ℜ c(|UArr|) =
      op-ntcf (lim-Obj c(|UArr|)) ∘_{NTCF−CF} inv-cf (op-cf-obj-comma ℜ c)
  unfolding op-ua-def ua-field-simps by (simp-all add: nat-omega-simps)
```

14.3.2 Opposite universal arrow for Kan extensions is a limit

lemma *op-ua-UArr-is-cat-limit*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $u : \mathfrak{T} \circ_{CF} \mathfrak{K}\ {}_{CF}\!\sqcap_{O}\ c >_{CF.colim} r : \mathfrak{K}\ {}_{CF}\!\downarrow\ c \mapsto\mapsto_{C\alpha} \mathfrak{A}$
shows *op-ntcf u* $\circ_{NTCF-CF}$ *inv-cf* (*op-cf-obj-comma* $\mathfrak{K}$ *c*) :
$r <_{CF.lim}$ *op-cf* $\mathfrak{T} \circ_{CF} c\ {}_{O}\!\sqcap_{CF}$ (*op-cf* $\mathfrak{K}$) : $c \downarrow_{CF}$ (*op-cf* $\mathfrak{K}$) $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{A}$
proof–

note [*cf-cs-simps*] = *is-iso-functor-is-iso-arr*(*2*,*3*)

let *?op-*$\mathfrak{K}$ = ‹λ*c. op-cf-obj-comma* $\mathfrak{K}$ *c*›
let *?op-*$\mathfrak{K}$*c* = ‹*?op-*$\mathfrak{K}$ *c*›
and *?op-ua-UArr* = ‹*op-ntcf u* $\circ_{NTCF-CF}$ *inv-cf* (*op-cf-obj-comma* $\mathfrak{K}$ *c*)›

interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*2*))
interpret *u*: *is-cat-colimit* α ‹$\mathfrak{K}$ ${}_{CF}\!\downarrow$ *c*› $\mathfrak{A}$ ‹$\mathfrak{T} \circ_{CF} \mathfrak{K}\ {}_{CF}\!\sqcap_{O}$ *c*› *r u*
by (*rule assms*(*4*))

from $\mathfrak{K}$.*op-cf-cf-obj-comma-proj*[*OF assms*(*3*)] **have**
op-cf ($\mathfrak{K}\ {}_{CF}\!\sqcap_{O}$ *c*) $\circ_{CF}$ *inv-cf* (*?op-*$\mathfrak{K}$ *c*) =
c ${}_{O}\!\sqcap_{CF}$ (*op-cf* $\mathfrak{K}$) $\circ_{CF}$ (*?op-*$\mathfrak{K}$ *c*) $\circ_{CF}$ *inv-cf* (*?op-*$\mathfrak{K}$ *c*)
by *simp*
from *this assms*(*3*) **have** [*cat-comma-cs-simps*]:
op-cf ($\mathfrak{K}\ {}_{CF}\!\sqcap_{O}$ *c*) $\circ_{CF}$ *inv-cf* (*?op-*$\mathfrak{K}$ *c*) = *c* ${}_{O}\!\sqcap_{CF}$ (*op-cf* $\mathfrak{K}$)
by
(
cs-prems
cs-simp: *cat-cs-simps cat-comma-cs-simps cf-cs-simps cat-op-simps*
cs-intro: *cf-cs-intros cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
from *assms*(*3*) **show** *?op-ua-UArr* :
$r <_{CF.lim}$ *op-cf* $\mathfrak{T} \circ_{CF} c\ {}_{O}\!\sqcap_{CF}$ (*op-cf* $\mathfrak{K}$) : $c \downarrow_{CF}$ (*op-cf* $\mathfrak{K}$) $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{A}$
by
(
cs-concl
cs-simp:
cf-cs-simps cat-cs-simps cat-comma-cs-simps cat-op-simps
$\mathfrak{K}$.*op-cf-cf-obj-comma-proj*[*symmetric*]
cs-intro:
cat-cs-intros
cf-cs-intros
cat-lim-cs-intros
cat-comma-cs-intros
cat-op-intros
)

qed

context
fixes *lim-Obj* :: *V* ⇒ *V* **and** *c* :: *V*
begin

lemmas *op-ua-UArr-is-cat-limit′* = *op-ua-UArr-is-cat-limit*
[

unfolded op-ua-components(*2*)[*symmetric*],
where *u*=‹*lim-Obj c*(|*UArr*|)› **and** *r*=‹*lim-Obj c*(|*UObj*|)› **and** *c*=*c*,
folded op-ua-components(*2*)[**where** *lim-Obj*=*lim-Obj* **and** *c*=*c*]
]

end

14.4 The Kan extension

The following subsection is based on the statement and proof of Theorem 1 in Chapter X-3 in [9].

14.4.1 Definition and elementary properties

definition *the-cf-rKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *the-cf-rKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj* =
[
($\lambda c \in_\circ \mathfrak{K}(|HomCod|)(|Obj|).$ *lim-Obj c*(|*UObj*|)),
(
$\lambda g \in_\circ \mathfrak{K}(|HomCod|)(|Arr|).$ *THE f*.
f :
lim-Obj ($\mathfrak{K}(|HomCod|)(|Dom|)(|g|)$)(|*UObj*|) $\mapsto_{\mathfrak{T}(|HomCod|)}$
lim-Obj ($\mathfrak{K}(|HomCod|)(|Cod|)(|g|)$)(|*UObj*|) $\wedge$
lim-Obj ($\mathfrak{K}(|HomCod|)(|Dom|)(|g|)$)(|*UArr*|) $\circ_{NTCF-CF}$ *g* $_A\downarrow_{CF}$ $\mathfrak{K}$ =
lim-Obj ($\mathfrak{K}(|HomCod|)(|Cod|)(|g|)$)(|*UArr*|) $\cdot_{NTCF}$
ntcf-const (($\mathfrak{K}(|HomCod|)(|Cod|)(|g|)$) $\downarrow_{CF}$ $\mathfrak{K}$) ($\mathfrak{T}(|HomCod|)$) *f*
),
$\mathfrak{K}(|HomCod|)$,
$\mathfrak{T}(|HomCod|)$
]$_\circ$

definition *the-ntcf-rKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *the-ntcf-rKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj* =
[
(
$\lambda c \in_\circ \mathfrak{T}(|HomDom|)(|Obj|).$
lim-Obj ($\mathfrak{K}(|ObjMap|)(|c|)$)(|*UArr*|)(|*NTMap*|)(|*0*, *c*, $\mathfrak{K}(|HomCod|)(|CId|)(\mathfrak{K}(|ObjMap|)(|c|))$|).
),
the-cf-rKe α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj* $\circ_{CF}$ $\mathfrak{K}$,
$\mathfrak{T}$,
$\mathfrak{T}(|HomDom|)$,
$\mathfrak{T}(|HomCod|)$
]$_\circ$

definition *the-cf-lKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *the-cf-lKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj* =
op-cf (*the-cf-rKe* α (*op-cf* $\mathfrak{T}$) (*op-cf* $\mathfrak{K}$) (*op-ua lim-Obj* $\mathfrak{K}$))

definition *the-ntcf-lKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *the-ntcf-lKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj* =
op-ntcf (*the-ntcf-rKe* α (*op-cf* $\mathfrak{T}$) (*op-cf* $\mathfrak{K}$) (*op-ua lim-Obj* $\mathfrak{K}$))

Components.

lemma *the-cf-rKe-components*:
shows *the-cf-rKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj*(|*ObjMap*|) =
($\lambda c \in_\circ \mathfrak{K}(|HomCod|)(|Obj|).$ *lim-Obj c*(|*UObj*|))
and *the-cf-rKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj*(|*ArrMap*|) =

(
$\lambda g \in_\circ \mathfrak{K}(\!|HomCod|\!)(\!|Arr|\!).\ THE\ f.$
$f :$
$lim\text{-}Obj\ (\mathfrak{K}(\!|HomCod|\!)(\!|Dom|\!)(\!|g|\!))(\!|UObj|\!) \mapsto_{\mathfrak{T}(\!|HomCod|\!)}$
$lim\text{-}Obj\ (\mathfrak{K}(\!|HomCod|\!)(\!|Cod|\!)(\!|g|\!))(\!|UObj|\!)\ \wedge$
$lim\text{-}Obj\ (\mathfrak{K}(\!|HomCod|\!)(\!|Dom|\!)(\!|g|\!))(\!|UArr|\!) \circ_{NTCF-CF}\ g\ _A\!\downarrow_{CF}\ \mathfrak{K} =$
$lim\text{-}Obj\ (\mathfrak{K}(\!|HomCod|\!)(\!|Cod|\!)(\!|g|\!))(\!|UArr|\!) \cdot_{NTCF}$
$ntcf\text{-}const\ ((\mathfrak{K}(\!|HomCod|\!)(\!|Cod|\!)(\!|g|\!))\downarrow_{CF}\ \mathfrak{K})\ (\mathfrak{T}(\!|HomCod|\!))\ f$
)
and *the-cf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|HomDom|\!) = \mathfrak{K}(\!|HomCod|\!)$
and *the-cf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|HomCod|\!) = \mathfrak{T}(\!|HomCod|\!)$
unfolding *the-cf-rKe-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *the-ntcf-rKe-components*:
shows *the-ntcf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|NTMap|\!) =$
(
$\lambda c \in_\circ \mathfrak{T}(\!|HomDom|\!)(\!|Obj|\!).$
$lim\text{-}Obj\ (\mathfrak{K}(\!|ObjMap|\!)(\!|c|\!))(\!|UArr|\!)(\!|NTMap|\!)(\!|0,\ c,\ \mathfrak{K}(\!|HomCod|\!)(\!|CId|\!)(\mathfrak{K}(\!|ObjMap|\!)(\!|c|\!))|\!)_\bullet$
)
and *the-ntcf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|NTDom|\!) =$ *the-cf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj \circ_{CF} \mathfrak{K}$
and *the-ntcf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|NTCod|\!) = \mathfrak{T}$
and *the-ntcf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|NTDGDom|\!) = \mathfrak{T}(\!|HomDom|\!)$
and *the-ntcf-rKe* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ lim\text{-}Obj(\!|NTDGCod|\!) = \mathfrak{T}(\!|HomCod|\!)$
unfolding *the-ntcf-rKe-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}\ \mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)
interpretation $\mathfrak{T}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *the-cf-rKe-components'* = *the-cf-rKe-components*[
where $\mathfrak{K}$=$\mathfrak{K}$ **and** $\mathfrak{T}$=$\mathfrak{T}$ **and** α=α, *unfolded* $\mathfrak{K}$.*cf-HomCod* $\mathfrak{T}$.*cf-HomCod*
]

lemmas [*cat-Kan-cs-simps*] = *the-cf-rKe-components'*(*3,4*)

lemmas *the-ntcf-rKe-components'* = *the-ntcf-rKe-components*[
where $\mathfrak{K}$=$\mathfrak{K}$ **and** $\mathfrak{T}$=$\mathfrak{T}$ **and** α=α, *unfolded* $\mathfrak{K}$.*cf-HomCod* $\mathfrak{T}$.*cf-HomCod* $\mathfrak{T}$.*cf-HomDom*
]

lemmas [*cat-Kan-cs-simps*] = *the-ntcf-rKe-components'*(*2–5*)

end

14.4.2 Functor: object map

mk-VLambda *the-cf-rKe-components*(*1*)
|*vsv the-cf-rKe-ObjMap-vsv*[*cat-Kan-cs-intros*]|

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}\ \mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)

mk-VLambda *the-cf-rKe-components'*(*1*)[*OF* $\mathfrak{K}$ $\mathfrak{T}$]
|*vdomain the-cf-rKe-ObjMap-vdomain*[*cat-Kan-cs-simps*]|
|*app the-cf-rKe-ObjMap-impl-app*[*cat-Kan-cs-simps*]|

lemma *the-cf-rKe-ObjMap-vrange*:
assumes $\bigwedge c.\ c \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow lim\text{-}Obj\ c(\!|UObj|\!) \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\mathcal{R}_\circ$ (*the-cf-rKe* α $\mathfrak{T}$ $\mathfrak{K}$ *lim-Obj*(\|*ObjMap*\|)) $\subseteq_\circ \mathfrak{A}(\!|Obj|\!)$
unfolding *the-cf-rKe-components'*[*OF* $\mathfrak{K}$ $\mathfrak{T}$]
by (*intro vrange-VLambda-vsubset assms*)

end

14.4.3 Functor: arrow map

mk-VLambda *the-cf-rKe-components*(*2*)
|*vsv the-cf-rKe-ArrMap-vsv*[*cat-Kan-cs-intros*]|

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)

mk-VLambda *the-cf-rKe-components*(*2*)[**where** α=α **and** $\mathfrak{K}$=$\mathfrak{K}$, *unfolded* $\mathfrak{K}$.*cf-HomCod*]
|*vdomain the-cf-rKe-ArrMap-vdomain*[*cat-Kan-cs-simps*]|

context
fixes $\mathfrak{A}$ $\mathfrak{T}$ c c' g
assumes $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and g: $g : c \mapsto_{\mathfrak{C}} c'$
begin

interpretation $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemma g': $g \in_\circ \mathfrak{C}(\!|Arr|\!)$ **using** g **by** *auto*

mk-VLambda *the-cf-rKe-components*(*2*)[
where α=α **and** $\mathfrak{K}$=$\mathfrak{K}$ **and** $\mathfrak{T}$=$\mathfrak{T}$, *unfolded* $\mathfrak{K}$.*cf-HomCod* $\mathfrak{T}$.*cf-HomCod*
]
|*app the-cf-rKe-ArrMap-app-impl'*|

lemmas *the-cf-rKe-ArrMap-app'* = *the-cf-rKe-ArrMap-app-impl'*[
OF g', *unfolded* $\mathfrak{K}$.*HomCod.cat-is-arrD*[*OF* g]
]

end

end

lemma *the-cf-rKe-ArrMap-app-impl*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $g : c \mapsto_{\mathfrak{C}} c'$
and $u : r <_{CF.lim} \mathfrak{T} \circ_{CF} c\ {}_O\!\bigsqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

```
    and u' : r' <_{CF.lim} 𝔗 ∘_{CF} c' _O∏_{CF} 𝔎 : c' ↓_{CF} 𝔎 ↦↦_{Cα} 𝔄
  shows ∃!f.
    f : r ↦_𝔄 r' ∧
    u ∘_{NTCF-CF} g _A↓_{CF} 𝔎 = u' ⋆_{NTCF} ntcf-const (c' ↓_{CF} 𝔎) 𝔄 f
proof–

  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  interpret u: is-cat-limit α ‹c ↓_{CF} 𝔎› 𝔄 ‹𝔗 ∘_{CF} c _O∏_{CF} 𝔎› r u
    by (rule assms(4))
  interpret u': is-cat-limit α ‹c' ↓_{CF} 𝔎› 𝔄 ‹𝔗 ∘_{CF} c' _O∏_{CF} 𝔎› r' u'
    by (rule assms(5))

  have const-r-def:
    cf-const (c' ↓_{CF} 𝔎) 𝔄 r = cf-const (c ↓_{CF} 𝔎) 𝔄 r ∘_{CF} g _A↓_{CF} 𝔎
  proof(rule cf-eqI)
    show const-r: cf-const (c' ↓_{CF} 𝔎) 𝔄 r : c' ↓_{CF} 𝔎 ↦↦_{Cα} 𝔄
      by (cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros)
    from assms(3) show const-r-g𝔎:
      cf-const (c ↓_{CF} 𝔎) 𝔄 r ∘_{CF} g _A↓_{CF} 𝔎 : c' ↓_{CF} 𝔎 ↦↦_{Cα} 𝔄
      by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)
    have ObjMap-dom-lhs: 𝒟∘ (cf-const (c' ↓_{CF} 𝔎) 𝔄 r(⁅ObjMap⁆)) = c' ↓_{CF} 𝔎(⁅Obj⁆)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3) have ObjMap-dom-rhs:
      𝒟∘ ((cf-const (c ↓_{CF} 𝔎) 𝔄 r ∘_{CF} g _A↓_{CF} 𝔎)(⁅ObjMap⁆)) = c' ↓_{CF} 𝔎(⁅Obj⁆)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
        )
    have ArrMap-dom-lhs: 𝒟∘ (cf-const (c' ↓_{CF} 𝔎) 𝔄 r(⁅ArrMap⁆)) = c' ↓_{CF} 𝔎(⁅Arr⁆)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3) have ArrMap-dom-rhs:
      𝒟∘ ((cf-const (c ↓_{CF} 𝔎) 𝔄 r ∘_{CF} g _A↓_{CF} 𝔎)(⁅ArrMap⁆)) = c' ↓_{CF} 𝔎(⁅Arr⁆)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
        )
    show
      cf-const (c' ↓_{CF} 𝔎) 𝔄 r(⁅ObjMap⁆) =
        (cf-const (c ↓_{CF} 𝔎) 𝔄 r ∘_{CF} g _A↓_{CF} 𝔎)(⁅ObjMap⁆)
    proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
      fix A assume prems: A ∈∘ c' ↓_{CF} 𝔎(⁅Obj⁆)
      from prems assms obtain b f
        where A-def: A = [0, b, f]∘
          and b: b ∈∘ 𝔅(⁅Obj⁆)
          and f: f : c' ↦_ℭ 𝔎(⁅ObjMap⁆)(⁅b⁆)
        by auto
      from assms(1,3) prems f b show
        cf-const (c' ↓_{CF} 𝔎) 𝔄 r(⁅ObjMap⁆)(⁅A⁆) =
          (cf-const (c ↓_{CF} 𝔎) 𝔄 r ∘_{CF} g _A↓_{CF} 𝔎)(⁅ObjMap⁆)(⁅A⁆)
        unfolding A-def
        by
          (
            cs-concl
```

```
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
      )
  qed
    (
      use assms(3) in
        ‹cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros›
    )+
  show
    cf-const (c' ↓CF ℜ) 𝔄 r(|ArrMap|) =
      (cf-const (c ↓CF ℜ) 𝔄 r ∘CF g A↓CF ℜ)(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    show vsv (cf-const (c' ↓CF ℜ) 𝔄 r(|ArrMap|))
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3) show vsv ((cf-const (c ↓CF ℜ) 𝔄 r ∘CF g A↓CF ℜ)(|ArrMap|))
      by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
    fix F assume prems: F ∈∘ c' ↓CF ℜ(|Arr|)
    with prems obtain A B where F: F : A ↦c' ↓CF ℜ B
      by (auto intro: is-arrI)
    with assms obtain b f b' f' h'
      where F-def: F = [[0, b, f]∘, [0, b', f']∘, [0, h']∘]∘
        and A-def: A = [0, b, f]∘
        and B-def: B = [0, b', f']∘
        and h': h' : b ↦𝔅 b'
        and f: f : c' ↦ℭ ℜ(|ObjMap|)(|b|)
        and f': f' : c' ↦ℭ ℜ(|ObjMap|)(|b'|)
        and f'-def: ℜ(|ArrMap|)(|h'|) ∘Aℭ f = f'
      by auto
    from prems assms(3) F g' h' f f' show
      cf-const (c' ↓CF ℜ) 𝔄 r(|ArrMap|)(|F|) =
        (cf-const (c ↓CF ℜ) 𝔄 r ∘CF g A↓CF ℜ)(|ArrMap|)(|F|)
      unfolding F-def A-def B-def
      by
        (
          cs-concl
            cs-simp: cat-comma-cs-simps cat-cs-simps f'-def[symmetric]
            cs-intro: cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
        )
  qed simp
qed simp-all

have 𝔗c'ℜ: 𝔗 ∘CF c' O⨅CF ℜ = 𝔗 ∘CF c O⨅CF ℜ ∘CF g A↓CF ℜ
proof(rule cf-eqI)
  show 𝔗 ∘CF c' O⨅CF ℜ : c' ↓CF ℜ ↦↦Cα 𝔄
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from assms show 𝔗 ∘CF c O⨅CF ℜ ∘CF g A↓CF ℜ : c' ↓CF ℜ ↦↦Cα 𝔄
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-comma-cs-intros cat-cs-intros
      )
  have ObjMap-dom-lhs: 𝒟∘ ((𝔗 ∘CF c' O⨅CF ℜ)(|ObjMap|)) = c' ↓CF ℜ(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms have ObjMap-dom-rhs:
    𝒟∘ ((𝔗 ∘CF c O⨅CF ℜ ∘CF g A↓CF ℜ)(|ObjMap|)) = c' ↓CF ℜ(|Obj|)
    by
      (
```

```
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-comma-cs-intros cat-cs-intros
    )
show (𝔗 ∘CF c′ OO⨅CF ℜ)(|ObjMap|) = (𝔗 ∘CF c OO⨅CF ℜ ∘CF g AA↓CF ℜ)(|ObjMap|)
proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
  from assms show vsv ((𝔗 ∘CF c′ OO⨅CF ℜ)(|ObjMap|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  from assms show vsv ((𝔗 ∘CF c OO⨅CF ℜ ∘CF g AA↓CF ℜ)(|ObjMap|))
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
  fix A assume prems: A ∈∘ c′ ↓CF ℜ(|Obj|)
  from assms(3) prems obtain b f
    where A-def: A = [0, b, f]∘
      and b: b ∈∘ 𝔅(|Obj|)
      and f: f : c′ ↦ℭ ℜ(|ObjMap|)(|b|)
    by auto
  from prems assms b f show
    (𝔗 ∘CF c′ OO⨅CF ℜ)(|ObjMap|)(|A|) =
      (𝔗 ∘CF c OO⨅CF ℜ ∘CF g AA↓CF ℜ)(|ObjMap|)(|A|)
    unfolding A-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed simp

have ArrMap-dom-lhs: 𝒟∘ ((𝔗 ∘CF c′ OO⨅CF ℜ)(|ArrMap|)) = c′ ↓CF ℜ(|Arr|)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
from assms have ArrMap-dom-rhs:
  𝒟∘ ((𝔗 ∘CF c OO⨅CF ℜ ∘CF g AA↓CF ℜ)(|ArrMap|)) = c′ ↓CF ℜ(|Arr|)
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-comma-cs-intros cat-cs-intros
    )

show (𝔗 ∘CF c′ OO⨅CF ℜ)(|ArrMap|) = (𝔗 ∘CF c OO⨅CF ℜ ∘CF g AA↓CF ℜ)(|ArrMap|)
proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
  from assms show vsv ((𝔗 ∘CF c′ OO⨅CF ℜ)(|ArrMap|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  from assms show vsv ((𝔗 ∘CF c OO⨅CF ℜ ∘CF g AA↓CF ℜ)(|ArrMap|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cs-intro: cat-cs-intros cat-comma-cs-intros
```

```
        )

      fix F assume prems: F ∈∘ c' ↓CF ℜ(|Arr|)
      with prems obtain A B where F: F : A ↦_{c' ↓CF ℜ} B
        unfolding cat-comma-cs-simps by (auto intro: is-arrI)
      with assms(3) obtain b f b' f' h'
        where F-def: F = [[0, b, f]∘, [0, b', f']∘, [0, h']∘]∘
          and A-def: A = [0, b, f]∘
          and B-def: B = [0, b', f']∘
          and h': h' : b ↦_𝔅 b'
          and f: f : c' ↦_ℭ ℜ(|ObjMap|)(|b|)
          and f': f' : c' ↦_ℭ ℜ(|ObjMap|)(|b'|)
          and f'-def: ℜ(|ArrMap|)(|h'|) ∘_{Aℭ} f = f'
        by auto
      from prems assms(3) F g' h' f f' show
        (𝔗 ∘CF c' O⨅CF ℜ)(|ArrMap|)(|F|) =
          (𝔗 ∘CF c O⨅CF ℜ ∘CF g A↓CF ℜ)(|ArrMap|)(|F|)
        unfolding F-def A-def B-def
        by
          (
            cs-concl
              cs-simp: cat-comma-cs-simps cat-cs-simps f'-def[symmetric]
              cs-intro: cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
          )
    qed simp
  qed simp-all

  from assms(1–3) have
    u ∘NTCF-CF g A↓CF ℜ : r <_{CF.cone} 𝔗 ∘CF c' O⨅CF ℜ : c' ↓CF ℜ ↦↦_{Cα} 𝔄
    by (intro is-cat-coneI)
      (
        cs-concl
          cs-intro: cat-cs-intros cat-comma-cs-intros cat-lim-cs-intros
          cs-simp: const-r-def 𝔗c'ℜ
      )+
  with u'.cat-lim-ua-fo show
    ∃!G.
      G : r ↦_𝔄 r' ∧
      u ∘NTCF-CF g A↓CF ℜ = u' ∙NTCF ntcf-const (c' ↓CF ℜ) 𝔄 G
    by simp

qed

lemma the-cf-rKe-ArrMap-app:
  assumes ℜ : 𝔅 ↦↦_{Cα} ℭ
    and 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
    and g : c ↦_ℭ c'
    and lim-Obj c(|UArr|) :
      lim-Obj c(|UObj|) <_{CF.lim} 𝔗 ∘CF c O⨅CF ℜ : c ↓CF ℜ ↦↦_{Cα} 𝔄
    and lim-Obj c'(|UArr|) :
      lim-Obj c'(|UObj|) <_{CF.lim} 𝔗 ∘CF c' O⨅CF ℜ : c' ↓CF ℜ ↦↦_{Cα} 𝔄
  shows the-cf-rKe α 𝔗 ℜ lim-Obj(|ArrMap|)(|g|) :
    lim-Obj c(|UObj|) ↦_𝔄 lim-Obj c'(|UObj|)
    and
      lim-Obj c(|UArr|) ∘NTCF-CF g A↓CF ℜ =
        lim-Obj c'(|UArr|) ∙NTCF
          ntcf-const (c' ↓CF ℜ) 𝔄 (the-cf-rKe α 𝔗 ℜ lim-Obj(|ArrMap|)(|g|))
    and
```

```
    ⟦
      f : lim-Obj c(|UObj|) ↦_𝔄 lim-Obj c'(|UObj|);
      lim-Obj c(|UArr|) ∘_NTCF-CF g _A↓_CF ℜ =
        lim-Obj c'(|UArr|) ·_NTCF ntcf-const (c' ↓_CF ℜ) 𝔄 f
    ⟧ ⟹ f = the-cf-rKe α 𝔗 ℜ lim-Obj(|ArrMap|)(|g|)
proof–

  interpret ℜ: is-functor α 𝔅 ℭ ℜ by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  interpret u: is-cat-limit
    α ‹c ↓_CF ℜ› 𝔄 ‹𝔗 ∘_CF c _O∏_CF ℜ› ‹lim-Obj c(|UObj|)› ‹lim-Obj c(|UArr|)›
    by (rule assms(4))
  interpret u': is-cat-limit
    α ‹c' ↓_CF ℜ› 𝔄 ‹𝔗 ∘_CF c' _O∏_CF ℜ› ‹lim-Obj c'(|UObj|)› ‹lim-Obj c'(|UArr|)›
    by (rule assms(5))

  from assms(3) have c: c ∈_∘ ℭ(|Obj|) and c': c' ∈_∘ ℭ(|Obj|) by auto

  note the-cf-rKe-ArrMap-app-impl' =
    the-cf-rKe-ArrMap-app-impl[OF assms]
  note the-f = theI'[OF the-cf-rKe-ArrMap-app-impl[OF assms]]
  note the-f-is-arr = the-f[THEN conjunct1]
    and the-f-commutes = the-f[THEN conjunct2]

  from assms(3) the-f-is-arr show
    the-cf-rKe α 𝔗 ℜ lim-Obj(|ArrMap|)(|g|) :
      lim-Obj c(|UObj|) ↦_𝔄 lim-Obj c'(|UObj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: the-cf-rKe-ArrMap-app' cs-intro: cat-cs-intros
      )
  moreover from assms(3) the-f-commutes show
    lim-Obj c(|UArr|) ∘_NTCF-CF g _A↓_CF ℜ =
      lim-Obj c'(|UArr|) ·_NTCF
        ntcf-const (c' ↓_CF ℜ) 𝔄 (the-cf-rKe α 𝔗 ℜ lim-Obj(|ArrMap|)(|g|))
    by
      (
        cs-concl cs-shallow
          cs-simp: the-cf-rKe-ArrMap-app' cs-intro: cat-cs-intros
      )
  ultimately show f = the-cf-rKe α 𝔗 ℜ lim-Obj(|ArrMap|)(|g|)
    if f : lim-Obj c(|UObj|) ↦_𝔄 lim-Obj c'(|UObj|)
      and lim-Obj c(|UArr|) ∘_NTCF-CF g _A↓_CF ℜ =
        lim-Obj c'(|UArr|) ·_NTCF ntcf-const (c' ↓_CF ℜ) 𝔄 f
    by (metis that the-cf-rKe-ArrMap-app-impl')

qed

lemma the-cf-rKe-ArrMap-is-arr'[cat-Kan-cs-intros]:
  assumes ℜ : 𝔅 ↦↦_Cα ℭ
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and g : c ↦_ℭ c'
    and lim-Obj c(|UArr|) :
      lim-Obj c(|UObj|) <_CF.lim 𝔗 ∘_CF c _O∏_CF ℜ : c ↓_CF ℜ ↦↦_Cα 𝔄
    and lim-Obj c'(|UArr|) :
      lim-Obj c'(|UObj|) <_CF.lim 𝔗 ∘_CF c' _O∏_CF ℜ : c' ↓_CF ℜ ↦↦_Cα 𝔄
    and a = lim-Obj c(|UObj|)
```

```
    and b = lim-Obj c'(|UObj|)
  shows the-cf-rKe α 𝔗 𝔎 lim-Obj(|ArrMap|)(|g|) : a ↦_𝔄 b
  unfolding assms(6,7) by (rule the-cf-rKe-ArrMap-app[OF assms(1–5)])

lemma lim-Obj-the-cf-rKe-commute:
  assumes 𝔎 : 𝔅 ↦↦_Cα ℭ
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and lim-Obj a(|UArr|) :
      lim-Obj a(|UObj|) <_CF.lim 𝔗 ∘_CF a _O⨅_CF 𝔎 : a ↓_CF 𝔎 ↦↦_Cα 𝔄
    and lim-Obj b(|UArr|) :
      lim-Obj b(|UObj|) <_CF.lim 𝔗 ∘_CF b _O⨅_CF 𝔎 : b ↓_CF 𝔎 ↦↦_Cα 𝔄
    and f : a ↦_ℭ b
    and [a', b', f']_∘ ∈_∘ b ↓_CF 𝔎(|Obj|)
  shows
    lim-Obj a(|UArr|)(|NTMap|)(|a', b', f' ∘_Aℭ f|)_• =
      lim-Obj b(|UArr|)(|NTMap|)(|a', b', f'|)_• ∘_A𝔄
        the-cf-rKe α 𝔗 𝔎 lim-Obj(|ArrMap|)(|f|)
proof–

  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))

  note f = 𝔎.HomCod.cat-is-arrD[OF assms(5)]

  interpret lim-a: is-cat-limit
    α ‹a ↓_CF 𝔎› 𝔄 ‹𝔗 ∘_CF a _O⨅_CF 𝔎› ‹lim-Obj a(|UObj|)› ‹lim-Obj a(|UArr|)›
    by (rule assms(3))
  interpret lim-b: is-cat-limit
    α ‹b ↓_CF 𝔎› 𝔄 ‹𝔗 ∘_CF b _O⨅_CF 𝔎› ‹lim-Obj b(|UObj|)› ‹lim-Obj b(|UArr|)›
    by (rule assms(4))

  note f-app = the-cf-rKe-ArrMap-app[
      where lim-Obj=lim-Obj, OF assms(1,2,5,3,4)
      ]

  from f-app(2) have lim-a-f𝔎-NTMap-app:
    (lim-Obj a(|UArr|) ∘_NTCF-CF f _A↓_CF 𝔎)(|NTMap|)(|A|) =
      (
        lim-Obj b(|UArr|) ·_NTCF
        ntcf-const (b ↓_CF 𝔎) 𝔄 (the-cf-rKe α 𝔗 𝔎 lim-Obj(|ArrMap|)(|f|))
      )(|NTMap|)(|A|)
    if ‹A ∈_∘ b ↓_CF 𝔎(|Obj|)› for A
    by simp
  show
    lim-Obj a(|UArr|)(|NTMap|)(|a', b', f' ∘_Aℭ f|)_• =
      lim-Obj b(|UArr|)(|NTMap|)(|a', b', f'|)_• ∘_A𝔄
        the-cf-rKe α 𝔗 𝔎 lim-Obj(|ArrMap|)(|f|)
  proof–
    from assms(5,6) have a'-def: a' = 0
      and b': b' ∈_∘ 𝔅(|Obj|)
      and f': f' : b ↦_ℭ 𝔎(|ObjMap|)(|b'|)
      by auto
    show
      lim-Obj a(|UArr|)(|NTMap|)(|a', b', f' ∘_Aℭ f|)_• =
        lim-Obj b(|UArr|)(|NTMap|)(|a', b', f'|)_• ∘_A𝔄
          the-cf-rKe α 𝔗 𝔎 lim-Obj(|ArrMap|)(|f|)
      using lim-a-f𝔎-NTMap-app[OF assms(6)] f' assms(3–6)
      unfolding a'-def
```

```
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros
      )
  qed

qed
```

14.4.4 Natural transformation: natural transformation map

```
mk-VLambda the-ntcf-rKe-components(1)
  |vsv the-ntcf-rKe-NTMap-vsv[cat-Kan-cs-intros]|

context
  fixes α 𝔄 𝔅 ℭ 𝔎 𝔗
  assumes 𝔎: 𝔎 : 𝔅 ↦↦_Cα ℭ
    and 𝔗: 𝔗 : 𝔅 ↦↦_Cα 𝔄
begin

interpretation 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule 𝔎)
interpretation 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule 𝔗)

mk-VLambda the-ntcf-rKe-components'(1)[OF 𝔎 𝔗]
  |vdomain the-ntcf-rKe-ObjMap-vdomain[cat-Kan-cs-simps]|
  |app the-ntcf-rKe-ObjMap-impl-app[cat-Kan-cs-simps]|

end
```

14.4.5 The Kan extension is a Kan extension

```
lemma the-cf-rKe-is-functor:
  assumes 𝔎 : 𝔅 ↦↦_Cα ℭ
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and ⋀c. c ∈_∘ ℭ(|Obj|) ⟹ lim-Obj c(|UArr|) :
      lim-Obj c(|UObj|) <_CF.lim 𝔗 ∘_CF c _O⨅_CF 𝔎 : c ↓_CF 𝔎 ↦↦_Cα 𝔄
  shows the-cf-rKe α 𝔗 𝔎 lim-Obj : ℭ ↦↦_Cα 𝔄
proof–

  let ?UObj = ‹λa. lim-Obj a(|UObj|)›
  let ?UArr = ‹λa. lim-Obj a(|UArr|)›
  let ?const-comma = ‹λa b. cf-const (a ↓_CF 𝔎) 𝔄 (?UObj b)›
  let ?the-cf-rKe = ‹the-cf-rKe α 𝔗 𝔎 lim-Obj›

  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))

  note [cat-lim-cs-intros] = is-cat-cone.cat-cone-obj

  show ?thesis
  proof(intro is-functorI')

    show vfsequence ?the-cf-rKe unfolding the-cf-rKe-def by simp
    show vcard ?the-cf-rKe = ℵ₄
      unfolding the-cf-rKe-def by (simp add: nat-omega-simps)
    show vsv (?the-cf-rKe(|ObjMap|))
      by (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)
```

moreover show $\mathcal{D}_\circ$ (*?the-cf-rKe*(|*ObjMap*|)) = $\mathfrak{C}$(|*Obj*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-Kan-cs-simps* **cs-intro**: *cat-cs-intros*)
moreover show $\mathcal{R}_\circ$ (*?the-cf-rKe*(|*ObjMap*|)) $\subseteq_\circ$ $\mathfrak{A}$(|*Obj*|)
proof
(
intro the-cf-rKe-ObjMap-vrange;
(*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)*?*
)
fix *c* **assume** *c* $\in_\circ$ $\mathfrak{C}$(|*Obj*|)
with *assms*(*3*)[*OF this*] **show** *?UObj c* $\in_\circ$ $\mathfrak{A}$(|*Obj*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-lim-cs-intros*)
qed
ultimately have [*cat-Kan-cs-intros*]:
?the-cf-rKe(|*ObjMap*|)(|*c*|) $\in_\circ$ $\mathfrak{A}$(|*Obj*|) **if** ‹*c* $\in_\circ$ $\mathfrak{C}$(|*Obj*|)› **for** *c*
by (*metis that vsubsetE vsv.vsv-value*)

show *?the-cf-rKe*(|*ArrMap*|)(|*f*|) :
?the-cf-rKe(|*ObjMap*|)(|*a*|) $\mapsto_{\mathfrak{A}}$ *?the-cf-rKe*(|*ObjMap*|)(|*b*|)
if *f* : *a* $\mapsto_{\mathfrak{C}}$ *b* **for** *a b f*
using *assms*(*2*) *that*
by
(
cs-concl
cs-simp: *cat-Kan-cs-simps*
cs-intro: *assms*(*3*) *cat-cs-intros cat-Kan-cs-intros*
)
then have [*cat-Kan-cs-intros*]: *?the-cf-rKe*(|*ArrMap*|)(|*f*|) : *A* $\mapsto_{\mathfrak{A}}$ *B*
if *A* = *?the-cf-rKe*(|*ObjMap*|)(|*a*|)
and *B* = *?the-cf-rKe*(|*ObjMap*|)(|*b*|)
and *f* : *a* $\mapsto_{\mathfrak{C}}$ *b*
for *A B a b f*
by (*simp add*: *that*)

show
?the-cf-rKe(|*ArrMap*|)(|*g* $\circ_{A\mathfrak{C}}$ *f*|) =
?the-cf-rKe(|*ArrMap*|)(|*g*|) $\circ_{A\mathfrak{A}}$ *?the-cf-rKe*(|*ArrMap*|)(|*f*|)
(**is** ‹*?the-cf-rKe*(|*ArrMap*|)(|*g* $\circ_{A\mathfrak{C}}$ *f*|) = *?the-rKe-g* $\circ_{A\mathfrak{A}}$ *?the-rKe-f*›)
if *g-is-arr*: *g* : *b* $\mapsto_{\mathfrak{C}}$ *c* **and** *f-is-arr*: *f* : *a* $\mapsto_{\mathfrak{C}}$ *b* **for** *b c g a f*
proof–

let *?ntcf-const-c* = ‹$\lambda f.$ *ntcf-const* (*c* $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$ *f*›

note *g* = $\mathfrak{K}$.*HomCod.cat-is-arrD*[*OF that*(*1*)]
and *f* = $\mathfrak{K}$.*HomCod.cat-is-arrD*[*OF that*(*2*)]
note *lim-a* = *assms*(*3*)[*OF f*(*2*)]
and *lim-b* = *assms*(*3*)[*OF g*(*2*)]
and *lim-c* = *assms*(*3*)[*OF g*(*3*)]
from *that* **have** *gf*: *g* $\circ_{A\mathfrak{C}}$ *f* : *a* $\mapsto_{\mathfrak{C}}$ *c*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

interpret *lim-a*: *is-cat-limit*
α ‹*a* $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ *a* $_O\bigsqcap_{CF}$ $\mathfrak{K}$› ‹*?UObj a*› ‹*?UArr a*›
by (*rule lim-a*)
interpret *lim-c*: *is-cat-limit*
α ‹*c* $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ *c* $_O\bigsqcap_{CF}$ $\mathfrak{K}$› ‹*?UObj c*› ‹*?UArr c*›
by (*rule lim-c*)

show *?thesis*

```
proof
  (
    rule sym,
    rule the-cf-rKe-ArrMap-app(3)[OF assms(1,2) gf lim-a lim-c]
  )

  from assms(1,2) that lim-a lim-b lim-c show
    ?the-rKe-g ∘_{A𝔄} ?the-rKe-f : ?UObj a ↦_𝔄 ?UObj c
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
      )

  show
    ?UArr a ∘_{NTCF-CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ =
      ?UArr c ⋆_{NTCF} ?ntcf-const-c (?the-rKe-g ∘_{A𝔄} ?the-rKe-f)
    (
      is
        ‹
          ?UArr a ∘_{NTCF-CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ =
            ?UArr c ⋆_{NTCF} ?ntcf-const-c ?the-rKe-gf
        ›
    )
  proof(rule ntcf-eqI)
    from that show
      ?UArr a ∘_{NTCF-CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ :
        cf-const (a ↓_{CF} ℜ) 𝔄 (?UObj a) ∘_{CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ ↦_{CF}
        𝔗 ∘_{CF} a _{O⨅CF} ℜ ∘_{CF} ((g ∘_{A𝔈} f) _{A↓CF} ℜ) :
        c ↓_{CF} ℜ ↦↦_{Cα} 𝔄
      by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
    have [cat-comma-cs-simps]:
      ?const-comma a a ∘_{CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ = ?const-comma c a
    proof(rule cf-eqI)
      from g-is-arr f-is-arr show
        ?const-comma a a ∘_{CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ : c ↓_{CF} ℜ ↦↦_{Cα} 𝔄
        by
          (
            cs-concl
              cs-simp: cat-comma-cs-simps cat-cs-simps
              cs-intro:
                cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
          )
      from g-is-arr f-is-arr show ?const-comma c a : c ↓_{CF} ℜ ↦↦_{Cα} 𝔄
        by
          (
            cs-concl
              cs-simp: cat-comma-cs-simps cat-cs-simps
              cs-intro:
                cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
          )
      from g-is-arr f-is-arr have ObjMap-dom-lhs:
        𝒟∘ ((?const-comma a a ∘_{CF} (g ∘_{A𝔈} f) _{A↓CF} ℜ)(‖ObjMap‖)) =
          c ↓_{CF} ℜ(‖Obj‖)
        by
          (
            cs-concl
              cs-simp: cat-comma-cs-simps cat-cs-simps
```

```
      cs-intro:
        cat-comma-cs-intros cat-lim-cs-intros cat-cs-intros
    )
from g-is-arr f-is-arr have ObjMap-dom-rhs:
  𝒟ₒ (?const-comma c a(|ObjMap|)) = c ↓CF ℜ(|Obj|)
  by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cat-cs-simps)

show
  (?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF ℜ)(|ObjMap|) =
    ?const-comma c a(|ObjMap|)
proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
  from f-is-arr g-is-arr show
    vsv ((?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF ℜ)(|ObjMap|))
    by
      (
        cs-concl
          cs-simp: cat-comma-cs-simps cat-cs-simps
          cs-intro:
            cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
      )
  fix A assume prems: A ∈ₒ c ↓CF ℜ(|Obj|)
  with g-is-arr obtain b′ f′
    where A-def: A = [0, b′, f′]ₒ
      and b′: b′ ∈ₒ 𝔅(|Obj|)
      and f′: f′ : c ↦𝔈 ℜ(|ObjMap|)(|b′|)
    by auto
  from prems b′ f′ g-is-arr f-is-arr show
    (?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF ℜ)(|ObjMap|)(|A|) =
      ?const-comma c a(|ObjMap|)(|A|)
    unfolding A-def
    by
      (
        cs-concl
          cs-simp: cat-comma-cs-simps cat-cs-simps
          cs-intro:
            cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
      )
qed (cs-concl cs-shallow cs-intro: cat-cs-intros)

from g-is-arr f-is-arr have ArrMap-dom-lhs:
  𝒟ₒ ((?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF ℜ)(|ArrMap|)) =
    c ↓CF ℜ(|Arr|)
  by
    (
      cs-concl
        cs-simp: cat-comma-cs-simps cat-cs-simps
        cs-intro:
          cat-comma-cs-intros cat-lim-cs-intros cat-cs-intros
    )
from g-is-arr f-is-arr have ArrMap-dom-rhs:
  𝒟ₒ (?const-comma c a(|ArrMap|)) = c ↓CF ℜ(|Arr|)
  by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cat-cs-simps)

show
  (?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF ℜ)(|ArrMap|) =
    ?const-comma c a(|ArrMap|)
proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
  from f-is-arr g-is-arr show
```

```
    vsv ((?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF 𝔎)(|ArrMap|))
    by
      (
        cs-concl
          cs-simp: cat-comma-cs-simps cat-cs-simps
          cs-intro:
            cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
      )
  fix F assume F ∈∘ c ↓CF 𝔎(|Arr|)
  then obtain A B where F: F : A ↦c ↓CF 𝔎 B
    unfolding cat-comma-cs-simps by (auto intro: is-arrI)
  with g-is-arr obtain b' f' b'' f'' h'
    where F-def: F = [[0, b', f']∘, [0, b'', f'']∘, [0, h']∘]∘
      and A-def: A = [0, b', f']∘
      and B-def: B = [0, b'', f'']∘
      and h': h' : b' ↦𝔅 b''
      and f': f' : c ↦𝔈 𝔎(|ObjMap|)(|b'|)
      and f'': f'' : c ↦𝔈 𝔎(|ObjMap|)(|b''|)
      and f''-def: 𝔎(|ArrMap|)(|h'|) ∘A𝔈 f' = f''
    by auto
  from F f-is-arr g-is-arr g' h' f' f'' show
    (?const-comma a a ∘CF (g ∘A𝔈 f) A↓CF 𝔎)(|ArrMap|)(|F|) =
      ?const-comma c a(|ArrMap|)(|F|)
    unfolding F-def A-def B-def
    by
      (
        cs-concl
          cs-intro:
            cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
          cs-simp:
            cat-cs-simps cat-comma-cs-simps f''-def[symmetric]
      )
qed (cs-concl cs-shallow cs-intro: cat-cs-intros)
qed simp-all

from that show
  ?UArr c ∙NTCF ?ntcf-const-c ?the-rKe-gf :
    cf-const (a ↓CF 𝔎) 𝔄 (?UObj a) ∘CF (g ∘A𝔈 f) A↓CF 𝔎 ↦CF
    𝔗 ∘CF a OO⨅CF 𝔎 ∘CF ((g ∘A𝔈 f) A↓CF 𝔎) :
    c ↓CF 𝔎 ↦↦Cα 𝔄
  by
    (
      cs-concl
        cs-simp: cat-Kan-cs-simps cat-comma-cs-simps cat-cs-simps
        cs-intro:
          cat-lim-cs-intros
          cat-comma-cs-intros
          cat-Kan-cs-intros
          cat-cs-intros
    )
from that have dom-lhs:
  𝒟∘ ((?UArr a ∘NTCF−CF (g ∘A𝔈 f) A↓CF 𝔎)(|NTMap|)) = c ↓CF 𝔎(|Obj|)
  by
    (
      cs-concl cs-shallow
        cs-intro: cat-cs-intros cat-comma-cs-intros
        cs-simp: cat-cs-simps cat-comma-cs-simps
    )
```

```
    from that have dom-rhs:
      𝒟∘ ((?UArr c ∙NTCF ?ntcf-const-c ?the-rKe-gf)(⌊NTMap⌋)) =
        c ↓CF ℜ(⌊Obj⌋)
      by
        (
          cs-concl
            cs-intro: cat-cs-intros cat-Kan-cs-intros cat-comma-cs-intros
            cs-simp: cat-Kan-cs-simps cat-cs-simps cat-comma-cs-simps
        )
    show
      (?UArr a ∘NTCF−CF (g ∘A𝔇 f) A↓CF ℜ)(⌊NTMap⌋) =
        (?UArr c ∙NTCF ?ntcf-const-c ?the-rKe-gf)(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix A assume prems: A ∈∘ c ↓CF ℜ(⌊Obj⌋)
      with g-is-arr obtain b′ f′
        where A-def: A = [0, b′, f′]∘
          and b′: b′ ∈∘ 𝔅(⌊Obj⌋)
          and f′: f′ : c ↦𝔇 ℜ(⌊ObjMap⌋)(⌊b′⌋)
        by auto
      note 𝔗.HomCod.cat-Comp-assoc[cat-cs-simps del]
        and ℜ.HomCod.cat-Comp-assoc[cat-cs-simps del]
        and category.cat-Comp-assoc[cat-cs-simps del]
      note [symmetric, cat-cs-simps] =
        lim-Obj-the-cf-rKe-commute[where lim-Obj=lim-Obj]
        ℜ.HomCod.cat-Comp-assoc
        𝔗.HomCod.cat-Comp-assoc
      from assms(1,2) that prems lim-a lim-b lim-c b′ f′ show
        (?UArr a ∘NTCF−CF (g ∘A𝔇 f) A↓CF ℜ)(⌊NTMap⌋)(⌊A⌋) =
          (?UArr c ∙NTCF ?ntcf-const-c ?the-rKe-gf)(⌊NTMap⌋)(⌊A⌋)
        unfolding A-def
        by
          (
            cs-concl
              cs-simp:
                cat-cs-simps cat-Kan-cs-simps cat-comma-cs-simps
              cs-intro:
                cat-cs-intros cat-Kan-cs-intros cat-comma-cs-intros
          )+
    qed (cs-concl cs-simp: cs-intro: cat-cs-intros)+
  qed simp-all
 qed
qed

show ?the-cf-rKe(⌊ArrMap⌋)(𝔇(⌊CId⌋)(⌊c⌋)) = 𝔄(⌊CId⌋)(⌊?the-cf-rKe(⌊ObjMap⌋)(⌊c⌋)⌋)
  if c ∈∘ 𝔇(⌊Obj⌋) for c
proof–

  let ?ntcf-const-c = ‹ntcf-const (c ↓CF ℜ) 𝔄 (𝔄(⌊CId⌋)(⌊?UObj c⌋))›

  note lim-c = assms(3)[OF that]

  from that have CId-c: 𝔇(⌊CId⌋)(⌊c⌋) : c ↦𝔇 c
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)

  interpret lim-c: is-cat-limit
    α ‹c ↓CF ℜ› 𝔄 ‹𝔗 ∘CF c O⨅CF ℜ› ‹?UObj c› ‹?UArr c›
    by (rule lim-c)
```

```
show ?thesis
proof
  (
    rule sym,
    rule the-cf-rKe-ArrMap-app(3)[
      where lim-Obj=lim-Obj, OF assms(1,2) CId-c lim-c lim-c
      ]
  )
  from that lim-c show
    𝔄(⁅CId⁆)(⁅?the-cf-rKe(⁅ObjMap⁆)(⁅c⁆)⁆) : ?UObj c ↦_𝔄 ?UObj c
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-lim-cs-intros
      )
  have ?UArr c ∘_NTCF-CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎 =  ?UArr c ⋆_NTCF ?ntcf-const-c
  proof(rule ntcf-eqI)
    from lim-c that show
      ?UArr c ∘_NTCF-CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎 :
        cf-const (c ↓_CF 𝔎) 𝔄 (?UObj c) ∘_CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎 ↦_CF
        𝔗 ∘_CF c _O⨅_CF 𝔎 ∘_CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎 :
        c ↓_CF 𝔎 ↦↦_Cα 𝔄
      by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
    from lim-c that show
      ?UArr c ⋆_NTCF ?ntcf-const-c :
        cf-const (c ↓_CF 𝔎) 𝔄 (?UObj c) ∘_CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎 ↦_CF
        𝔗 ∘_CF c _O⨅_CF 𝔎 ∘_CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎 :
        c ↓_CF 𝔎 ↦↦_Cα 𝔄
      by
        (
          cs-concl
            cs-intro: cat-cs-intros
            cs-simp: 𝔎.cf-arr-cf-comma-CId cat-cs-simps
            cs-intro: cat-lim-cs-intros
        )
    from that have dom-lhs:
      𝒟∘ ((?UArr c ∘_NTCF-CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎)(⁅NTMap⁆)) = c ↓_CF 𝔎(⁅Obj⁆)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )

    from that have dom-rhs:
      𝒟∘ ((?UArr c ⋆_NTCF ?ntcf-const-c)(⁅NTMap⁆)) = c ↓_CF 𝔎(⁅Obj⁆)
      by
        (
          cs-concl
            cs-intro: cat-lim-cs-intros cat-cs-intros
            cs-simp: cat-cs-simps
        )
    show
      (?UArr c ∘_NTCF-CF (ℭ(⁅CId⁆)(⁅c⁆)) _A↓_CF 𝔎)(⁅NTMap⁆) =
        (?UArr c ⋆_NTCF ?ntcf-const-c)(⁅NTMap⁆)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix A assume prems: A ∈∘ c ↓_CF 𝔎(⁅Obj⁆)
```

```
        with that obtain b f
          where A-def: A = [0, b, f]∘
            and b: b ∈∘ 𝔅(|Obj|)
            and f: f : c ↦ℭ ℜ(|ObjMap|)(|b|)
          by auto
        from that prems f have
          ?UArr c(|NTMap|)(|0, b, f|)• : ?UObj c ↦𝔄 𝔗(|ObjMap|)(|b|)
          unfolding A-def
          by
            (
              cs-concl
                cs-simp: cat-cs-simps cat-comma-cs-simps
                cs-intro: cat-comma-cs-intros cat-cs-intros
            )
        from that prems f show
          (?UArr c ∘NTCF-CF (ℭ(|CId|)(|c|)) A↓CF ℜ)(|NTMap|)(|A|) =
            (?UArr c •NTCF ?ntcf-const-c)(|NTMap|)(|A|)
          unfolding A-def
          by
            (
              cs-concl
                cs-simp: cat-cs-simps cat-comma-cs-simps
                cs-intro:
                  cat-lim-cs-intros cat-comma-cs-intros cat-cs-intros
            )
      qed (cs-concl cs-intro: cat-cs-intros)
    qed simp-all

    with that show
      ?UArr c ∘NTCF-CF (ℭ(|CId|)(|c|)) A↓CF ℜ =
        ?UArr c •NTCF ntcf-const (c ↓CF ℜ) 𝔄 (𝔄(|CId|)(|?the-cf-rKe(|ObjMap|)(|c|)|))
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
        )

  qed

 qed
  (
    cs-concl
      cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
  )+

qed

lemma the-cf-lKe-is-functor:
  assumes ℜ : 𝔅 ↦↦Cα ℭ
    and 𝔗 : 𝔅 ↦↦Cα 𝔄
    and ⋀c. c ∈∘ ℭ(|Obj|) ⟹ lim-Obj c(|UArr|) :
      𝔗 ∘CF ℜ CF↑O c >CF.colim lim-Obj c(|UObj|) : ℜ CF↓ c ↦↦Cα 𝔄
  shows the-cf-lKe α 𝔗 ℜ lim-Obj : ℭ ↦↦Cα 𝔄
proof–
  interpret ℜ: is-functor α 𝔅 ℭ ℜ by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
```

```
  {
    fix c assume prems: c ∈₀ ℭ⦇Obj⦈
    from assms(3)[OF this] have lim-Obj-UArr: lim-Obj c⦇UArr⦈ :
      𝔗 ∘_CF 𝔎 _CF↓ c >_CF.colim lim-Obj c⦇UObj⦈ : 𝔎 _CF↓ c ↦↦_Cα 𝔄.
    then interpret lim-Obj-c: is-cat-colimit
      α ‹𝔎 _CF↓ c› 𝔄 ‹𝔗 ∘_CF 𝔎 _CF↓_O c› ‹lim-Obj c⦇UObj⦈› ‹lim-Obj c⦇UArr⦈›
      by simp
    note op-ua-UArr-is-cat-limit'[
      where lim-Obj=lim-Obj, OF assms(1,2) prems lim-Obj-UArr
      ]
  }
  note the-cf-rKe-is-functor = the-cf-rKe-is-functor
    [
      OF 𝔎.is-functor-op 𝔗.is-functor-op,
      unfolded cat-op-simps,
      where lim-Obj=‹op-ua lim-Obj 𝔎›,
      unfolded cat-op-simps,
      OF this,
      simplified,
      folded the-cf-lKe-def
    ]
  show the-cf-lKe α 𝔗 𝔎 lim-Obj : ℭ ↦↦_Cα 𝔄
    by
      (
        rule is-functor.is-functor-op
          [
            OF the-cf-rKe-is-functor,
            folded the-cf-lKe-def,
            unfolded cat-op-simps
          ]
      )
qed

lemma the-ntcf-rKe-is-ntcf:
  assumes 𝔎 : 𝔅 ↦↦_Cα ℭ
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and ⋀c. c ∈₀ ℭ⦇Obj⦈ ⟹ lim-Obj c⦇UArr⦈ :
      lim-Obj c⦇UObj⦈ <_CF.lim 𝔗 ∘_CF c _O↓_CF 𝔎 : c ↓_CF 𝔎 ↦↦_Cα 𝔄
  shows the-ntcf-rKe α 𝔗 𝔎 lim-Obj :
    the-cf-rKe α 𝔗 𝔎 lim-Obj ∘_CF 𝔎 ↦_CF 𝔗 : 𝔅 ↦↦_Cα 𝔄
proof–

  let ?UObj = ‹λa. lim-Obj a⦇UObj⦈›
  let ?UArr = ‹λa. lim-Obj a⦇UArr⦈›
  let ?const-comma = ‹λa b. cf-const (a ↓_CF 𝔎) 𝔄 (?UObj b)›
  let ?the-cf-rKe = ‹the-cf-rKe α 𝔗 𝔎 lim-Obj›
  let ?the-ntcf-rKe = ‹the-ntcf-rKe α 𝔗 𝔎 lim-Obj›

  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  interpret cf-rKe: is-functor α ℭ 𝔄 ‹?the-cf-rKe›
    by (rule the-cf-rKe-is-functor[OF assms, simplified])

  show ?thesis
  proof(rule is-ntcfI')
    show vfsequence ?the-ntcf-rKe unfolding the-ntcf-rKe-def by simp
    show vcard ?the-ntcf-rKe = 5₍ℕ₎
      unfolding the-ntcf-rKe-def by (simp add: nat-omega-simps)
```

show *?the-ntcf-rKe*(|*NTMap*|)(|*b*|) :
(*?the-cf-rKe* $\circ_{CF}$ $\mathfrak{K}$)(|*ObjMap*|)(|*b*|) $\mapsto_{\mathfrak{A}}$ $\mathfrak{T}$(|*ObjMap*|)(|*b*|)
if $b \in_{\circ} \mathfrak{B}$(|*Obj*|) **for** *b*
proof–
let *?*$\mathfrak{K}b$ = ‹$\mathfrak{K}$(|*ObjMap*|)(|*b*|)›
from *that* **have** $\mathfrak{K}b$: $\mathfrak{K}$(|*ObjMap*|)(|*b*|) $\in_{\circ}$ $\mathfrak{C}$(|*Obj*|)
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
note *lim-*$\mathfrak{K}b$ = *assms*(*3*)[*OF* $\mathfrak{K}b$]
interpret *lim-*$\mathfrak{K}b$: *is-cat-limit*
α ‹*?*$\mathfrak{K}b$ $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ *?*$\mathfrak{K}b$ $_{O}\prod_{CF}$ $\mathfrak{K}$› ‹*?UObj ?*$\mathfrak{K}b$› ‹*?UArr ?*$\mathfrak{K}b$›
by (*rule lim-*$\mathfrak{K}b$)
from *that lim-*$\mathfrak{K}b$ **show** *?thesis*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros*
)+
qed
show
?the-ntcf-rKe(|*NTMap*|)(|*b*|) $\circ_{A\mathfrak{A}}$ (*?the-cf-rKe* $\circ_{CF}$ $\mathfrak{K}$)(|*ArrMap*|)(|*f*|) =
$\mathfrak{T}$(|*ArrMap*|)(|*f*|) $\circ_{A\mathfrak{A}}$ *?the-ntcf-rKe*(|*NTMap*|)(|*a*|)
if $f : a \mapsto_{\mathfrak{B}} b$ **for** *a b f*
proof–
let *?*$\mathfrak{K}a$ = ‹$\mathfrak{K}$(|*ObjMap*|)(|*a*|)› **and** *?*$\mathfrak{K}b$ = ‹$\mathfrak{K}$(|*ObjMap*|)(|*b*|)› **and** *?*$\mathfrak{K}f$ = ‹$\mathfrak{K}$(|*ArrMap*|)(|*f*|)›
from *that* **have** $\mathfrak{K}a$: *?*$\mathfrak{K}a \in_{\circ} \mathfrak{C}$(|*Obj*|)
and $\mathfrak{K}b$: *?*$\mathfrak{K}b \in_{\circ} \mathfrak{C}$(|*Obj*|)
and $\mathfrak{K}f$: *?*$\mathfrak{K}f$: *?*$\mathfrak{K}a \mapsto_{\mathfrak{C}}$ *?*$\mathfrak{K}b$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)+
note *lim-*$\mathfrak{K}a$ = *assms*(*3*)[*OF* $\mathfrak{K}a$]
and *lim-*$\mathfrak{K}b$ = *assms*(*3*)[*OF* $\mathfrak{K}b$]
from *that* **have** *z-b-*$\mathfrak{K}b$: [*0*, *b*, $\mathfrak{C}$(|*CId*|)(|*?*$\mathfrak{K}b$|)]$_{\circ} \in_{\circ}$ *?*$\mathfrak{K}b \downarrow_{CF}$ $\mathfrak{K}$(|*Obj*|)
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
from
lim-Obj-the-cf-rKe-commute[
OF assms(*1,2*) *lim-*$\mathfrak{K}a$ *lim-*$\mathfrak{K}b$ $\mathfrak{K}f$ *z-b-*$\mathfrak{K}b$, *symmetric*
]
that
have [*cat-Kan-cs-simps*]:
?UArr ?$\mathfrak{K}b$(|*NTMap*|)(|*0*, *b*, $\mathfrak{C}$(|*CId*|)(|*?*$\mathfrak{K}b$|)|)$_{\bullet}$ $\circ_{A\mathfrak{A}}$ *?the-cf-rKe*(|*ArrMap*|)(|*?*$\mathfrak{K}f$|) =
?UArr ?$\mathfrak{K}a$(|*NTMap*|)(|*0*, *b*, *?*$\mathfrak{K}f$|)$_{\bullet}$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
interpret *lim-*$\mathfrak{K}a$: *is-cat-limit*
α ‹*?*$\mathfrak{K}a$ $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ *?*$\mathfrak{K}a$ $_{O}\prod_{CF}$ $\mathfrak{K}$› ‹*?UObj ?*$\mathfrak{K}a$› ‹*?UArr ?*$\mathfrak{K}a$›
by (*rule lim-*$\mathfrak{K}a$)
interpret *lim-*$\mathfrak{K}b$: *is-cat-limit*
α ‹*?*$\mathfrak{K}b$ $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ *?*$\mathfrak{K}b$ $_{O}\prod_{CF}$ $\mathfrak{K}$› ‹*?UObj ?*$\mathfrak{K}b$› ‹*?UArr ?*$\mathfrak{K}b$›
by (*rule lim-*$\mathfrak{K}b$)
note *lim-*$\mathfrak{K}a$.*cat-cone-Comp-commute*[*cat-cs-simps del*]
note *lim-*$\mathfrak{K}b$.*cat-cone-Comp-commute*[*cat-cs-simps del*]
from *that* **have**
[[*0*, *a*, $\mathfrak{C}$(|*CId*|)(|*?*$\mathfrak{K}a$|)]$_{\circ}$, [*0*, *b*, *?*$\mathfrak{K}f$]$_{\circ}$, [*0*, *f*]$_{\circ}$]$_{\circ}$:
[*0*, *a*, $\mathfrak{C}$(|*CId*|)(|*?*$\mathfrak{K}a$|)]$_{\circ} \mapsto_{(?\mathfrak{K}a) \downarrow_{CF} \mathfrak{K}}$ [*0*, *b*, *?*$\mathfrak{K}f$]$_{\circ}$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*

```
      )
    from lim-𝔎a.ntcf-Comp-commute[OF this, symmetric] that
    have [cat-Kan-cs-simps]:
      𝔗(⁅ArrMap⁆)(⁅f⁆) ∘_{A𝔄} ?UArr (?𝔎a)(⁅NTMap⁆) (⁅0, a, ℭ(⁅CId⁆)(⁅?𝔎a⁆)⁆)• =
        ?UArr ?𝔎a(⁅NTMap⁆)(⁅0, b, ?𝔎f⁆)•
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cat-comma-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-1-is-arrI
        )
    from that show ?thesis
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
        )
  qed
qed
  (
    cs-concl
      cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
  )+

qed

lemma the-ntcf-lKe-is-ntcf:
  assumes 𝔎 : 𝔅 ↦↦_{Cα} ℭ
    and 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
    and ⋀c. c ∈₀ ℭ(⁅Obj⁆) ⟹ lim-Obj c(⁅UArr⁆) :
      𝔗 ∘_{CF} 𝔎 _{CF}↑_O c >_{CF.colim} lim-Obj c(⁅UObj⁆) : 𝔎 _{CF}↓ c ↦↦_{Cα} 𝔄
  shows the-ntcf-lKe α 𝔗 𝔎 lim-Obj :
    𝔗 ↦_{CF} the-cf-lKe α 𝔗 𝔎 lim-Obj ∘_{CF} 𝔎 : 𝔅 ↦↦_{Cα} 𝔄
proof–
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  {
    fix c assume prems: c ∈₀ ℭ(⁅Obj⁆)
    from assms(3)[OF this] have lim-Obj-UArr: lim-Obj c(⁅UArr⁆) :
      𝔗 ∘_{CF} 𝔎 _{CF}↑_O c >_{CF.colim} lim-Obj c(⁅UObj⁆) : 𝔎 _{CF}↓ c ↦↦_{Cα} 𝔄.
    then interpret lim-Obj-c: is-cat-colimit
      α ‹𝔎 _{CF}↓ c› 𝔄 ‹𝔗 ∘_{CF} 𝔎 _{CF}↑_O c› ‹lim-Obj c(⁅UObj⁆)› ‹lim-Obj c(⁅UArr⁆)›
      by simp
    note op-ua-UArr-is-cat-limit'[
      where lim-Obj=lim-Obj, OF assms(1,2) prems lim-Obj-UArr
      ]
  }
  note the-ntcf-rKe-is-ntcf = the-ntcf-rKe-is-ntcf
    [
      OF 𝔎.is-functor-op 𝔗.is-functor-op,
      unfolded cat-op-simps,
      where lim-Obj=‹op-ua lim-Obj 𝔎›,
      unfolded cat-op-simps,
      OF this,
      simplified
    ]
  show the-ntcf-lKe α 𝔗 𝔎 lim-Obj :
    𝔗 ↦_{CF} the-cf-lKe α 𝔗 𝔎 lim-Obj ∘_{CF} 𝔎 : 𝔅 ↦↦_{Cα} 𝔄
```

```
    by
      (
        rule is-ntcf.is-ntcf-op
          [
            OF the-ntcf-rKe-is-ntcf,
            unfolded cat-op-simps,
            folded the-cf-lKe-def the-ntcf-lKe-def
          ]
      )
qed

lemma the-ntcf-rKe-is-cat-rKe:
  assumes 𝔎 : 𝔅 ↦↦_Cα ℭ
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and ⋀c. c ∈₀ ℭ⦇Obj⦈ ⟹ lim-Obj c⦇UArr⦈ :
      lim-Obj c⦇UObj⦈ <_CF.lim 𝔗 ∘_CF c _O⨅_CF 𝔎 : c ↓_CF 𝔎 ↦↦_Cα 𝔄
  shows the-ntcf-rKe α 𝔗 𝔎 lim-Obj :
    the-cf-rKe α 𝔗 𝔎 lim-Obj ∘_CF 𝔎 ↦_CF.rKeα 𝔗 : 𝔅 ↦_C ℭ ↦_C 𝔄
proof–

  let ?UObj = ‹λa. lim-Obj a⦇UObj⦈›
  let ?UArr = ‹λa. lim-Obj a⦇UArr⦈›
  let ?the-cf-rKe = ‹the-cf-rKe α 𝔗 𝔎 lim-Obj›
  let ?the-ntcf-rKe = ‹the-ntcf-rKe α 𝔗 𝔎 lim-Obj›

  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  interpret cf-rKe: is-functor α ℭ 𝔄 ?the-cf-rKe
    by (rule the-cf-rKe-is-functor[OF assms, simplified])
  interpret ntcf-rKe: is-ntcf α 𝔅 𝔄 ‹?the-cf-rKe ∘_CF 𝔎› 𝔗 ?the-ntcf-rKe
    by (intro the-ntcf-rKe-is-ntcf assms(3))
      (cs-concl cs-shallow cs-intro: cat-cs-intros)+

  show ?thesis
  proof(rule is-cat-rKeI′)

    fix 𝔊 ε assume prems:
      𝔊 : ℭ ↦↦_Cα 𝔄 ε : 𝔊 ∘_CF 𝔎 ↦_CF 𝔗 : 𝔅 ↦↦_Cα 𝔄

    interpret 𝔊: is-functor α ℭ 𝔄 𝔊 by (rule prems(1))
    interpret ε: is-ntcf α 𝔅 𝔄 ‹𝔊 ∘_CF 𝔎› 𝔗 ε by (rule prems(2))

    define ε′ where ε′ c =
      [
        (λA∈₀c ↓_CF 𝔎⦇Obj⦈. ε⦇NTMap⦈⦇A⦇1_ℕ⦈⦈ ∘_A𝔄 𝔊⦇ArrMap⦈⦇A⦇2_ℕ⦈⦈),
        cf-const (c ↓_CF 𝔎) 𝔄 (𝔊⦇ObjMap⦈⦇c⦈),
        𝔗 ∘_CF c _O⨅_CF 𝔎,
        c ↓_CF 𝔎,
        𝔄
      ]₀
      for c

    have ε′-components:
      ε′ c⦇NTMap⦈ = (λA∈₀c ↓_CF 𝔎⦇Obj⦈. ε⦇NTMap⦈⦇A⦇1_ℕ⦈⦈ ∘_A𝔄 𝔊⦇ArrMap⦈⦇A⦇2_ℕ⦈⦈)
      ε′ c⦇NTDom⦈ = cf-const (c ↓_CF 𝔎) 𝔄 (𝔊⦇ObjMap⦈⦇c⦈)
      ε′ c⦇NTCod⦈ = 𝔗 ∘_CF c _O⨅_CF 𝔎
      ε′ c⦇NTDGDom⦈ = c ↓_CF 𝔎
      ε′ c⦇NTDGCod⦈ = 𝔄
```

```
  for c
  unfolding ε′-def nt-field-simps by (simp-all add: nat-omega-simps)
note [cat-Kan-cs-simps] = ε′-components(2–5)
have [cat-Kan-cs-simps]: ε′ c(|NTMap|)(|A|) = ε(|NTMap|)(|b|) ∘_{A𝔄} 𝔊(|ArrMap|)(|f|)
  if A = [a, b, f]∘ and [a, b, f]∘ ∈∘ c ↓_{CF} ℜ(|Obj|) for A a b c f
  using that unfolding ε′-components by (auto simp: nat-omega-simps)

have ε′: ε′ c : 𝔊(|ObjMap|)(|c|) <_{CF.cone} 𝔗 ∘_{CF} c _O⨅_{CF} ℜ : c ↓_{CF} ℜ ↦↦_{Cα} 𝔄
  and ε′-unique: ∃!f′.
    f′ : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c ∧
    ε′ c = ?UArr c ∙_{NTCF} ntcf-const (c ↓_{CF} ℜ) 𝔄 f′
  if c: c ∈∘ ℭ(|Obj|) for c
proof–
  from that have ?the-cf-rKe(|ObjMap|)(|c|) = ?UObj c
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
      )
  interpret lim-c: is-cat-limit
    α ‹c ↓_{CF} ℜ› 𝔄 ‹𝔗 ∘_{CF} c _O⨅_{CF} ℜ› ‹?UObj c› ‹?UArr c›
    by (rule assms(3)[OF that])
  show ε′ c : 𝔊(|ObjMap|)(|c|) <_{CF.cone} 𝔗 ∘_{CF} c _O⨅_{CF} ℜ : c ↓_{CF} ℜ ↦↦_{Cα} 𝔄
  proof(intro is-cat-coneI is-ntcfI′)
    show vfsequence (ε′ c) unfolding ε′-def by simp
    show vcard (ε′ c) = 5_ℕ unfolding ε′-def by (simp add: nat-omega-simps)
    show vsv (ε′ c(|NTMap|)) unfolding ε′-components by simp
    show 𝒟∘ (ε′ c(|NTMap|)) = c ↓_{CF} ℜ(|Obj|) unfolding ε′-components by simp
    show ε′ c(|NTMap|)(|A|) :
      cf-const (c ↓_{CF} ℜ) 𝔄 (𝔊(|ObjMap|)(|c|))(|ObjMap|)(|A|) ↦_𝔄
      (𝔗 ∘_{CF} c _O⨅_{CF} ℜ)(|ObjMap|)(|A|)
      if A ∈∘ c ↓_{CF} ℜ(|Obj|) for A
    proof–
      from that prems c obtain b f
        where A-def: A = [0, b, f]∘
          and b: b ∈∘ 𝔅(|Obj|)
          and f: f : c ↦_ℭ ℜ(|ObjMap|)(|b|)
        by auto
      from that prems f c that b f show ?thesis
        unfolding A-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cat-Kan-cs-simps cat-comma-cs-simps
              cs-intro: cat-cs-intros cat-comma-cs-intros
          )
    qed
    show
      ε′ c(|NTMap|)(|B|) ∘_{A𝔄} cf-const (c ↓_{CF} ℜ) 𝔄 (𝔊(|ObjMap|)(|c|))(|ArrMap|)(|F|) =
        (𝔗 ∘_{CF} c _O⨅_{CF} ℜ)(|ArrMap|)(|F|) ∘_{A𝔄} ε′ c(|NTMap|)(|A|)
      if F : A ↦_{c ↓_{CF} ℜ} B for A B F
    proof–
      from that c
      obtain b f b′ f′ k
        where F-def: F = [[0, b, f]∘, [0, b′, f′]∘, [0, k]∘]∘
          and A-def: A = [0, b, f]∘
          and B-def: B = [0, b′, f′]∘
          and k: k : b ↦_𝔅 b′
```

```
        and f: f : c ↦_ℭ ℜ(|ObjMap|)(|b|)
        and f'': f' : c ↦_ℭ ℜ(|ObjMap|)(|b'|)
        and f'-def: ℜ(|ArrMap|)(|k|) ∘_Aℭ f = f'
      by auto
    from c that k f f' show ?thesis
      unfolding F-def A-def B-def
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-comma-cs-simps
              cat-Kan-cs-simps
              ε.ntcf-Comp-commute''
              f'-def[symmetric]
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed
qed
  (
    use c that in
      ‹cs-concl cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros›
  )+
from is-cat-limit.cat-lim-ua-fo[OF assms(3)[OF that] this] show
  ∃!f'.
    f' : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c ∧
    ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 f'
  by simp
qed

define σ :: V where
  σ =
    [
      (
        λc∈₀ℭ(|Obj|). THE f.
          f : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c ∧
          ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 f
      ),
      𝔊,
      ?the-cf-rKe,
      ℭ,
      𝔄
    ]₀

have σ-components:
  σ(|NTMap|) =
    (
      λc∈₀ℭ(|Obj|). THE f.
        f : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c ∧
        ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 f
    )
  σ(|NTDom|) = 𝔊
  σ(|NTCod|) = ?the-cf-rKe
  σ(|NTDGDom|) = ℭ
  σ(|NTDGCod|) = 𝔄
  unfolding σ-def nt-field-simps by (simp-all add: nat-omega-simps)

note [cat-Kan-cs-simps] = σ-components(2–5)
```

```
have σ-NTMap-app-def: σ(|NTMap|)(|c|) =
  (
    THE f.
      f : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c ∧
      ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 f
  )
  if c ∈_∘ ℭ(|Obj|) for c
  using that unfolding σ-components by simp

have σ-NTMap-app-is-arr: σ(|NTMap|)(|c|) : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c
  and ε'-σ-commute:
    ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 (σ(|NTMap|)(|c|))
  and σ-NTMap-app-unique:
    ⟦
      f : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c;
      ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 f
    ⟧ ⟹ f = σ(|NTMap|)(|c|)
    if c: c ∈_∘ ℭ(|Obj|) for c f
proof–
  have
    σ(|NTMap|)(|c|) : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c ∧
    ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 (σ(|NTMap|)(|c|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps σ-NTMap-app-def
          cs-intro: theI' ε'-unique that
      )
  then show σ(|NTMap|)(|c|) : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c
    and ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 (σ(|NTMap|)(|c|))
    by simp-all
  with c ε'-unique[OF c] show f = σ(|NTMap|)(|c|)
    if f : 𝔊(|ObjMap|)(|c|) ↦_𝔄 ?UObj c
      and ε' c = ?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 f
    using that by metis
qed

have σ-NTMap-app-is-arr'[cat-Kan-cs-intros]: σ(|NTMap|)(|c|) : a ↦_𝔄' b
  if c ∈_∘ ℭ(|Obj|)
    and a = 𝔊(|ObjMap|)(|c|)
    and b = ?UObj c
    and 𝔄' = 𝔄
  for 𝔄' a b c
  by (simp add: that σ-NTMap-app-is-arr)

have ε'-NTMap-app-def:
  ε' c(|NTMap|)(|A|) =
    (?UArr c ∙_NTCF ntcf-const (c ↓_CF ℜ) 𝔄 (σ(|NTMap|)(|c|)))(|NTMap|)(|A|)
  if A ∈_∘ c ↓_CF ℜ(|Obj|) and c ∈_∘ ℭ(|Obj|) for A c
  using ε'-σ-commute[OF that(2)] by simp
have εb-𝔊f:
  ε(|NTMap|)(|b|) ∘_A𝔄 𝔊(|ArrMap|)(|f|) =
    ?UArr c(|NTMap|)(|a, b, f|)_• ∘_A𝔄 σ(|NTMap|)(|c|)
  if A = [a, b, f]_∘ and A ∈_∘ c ↓_CF ℜ(|Obj|) and c ∈_∘ ℭ(|Obj|)
  for A a b c f
proof–
  interpret lim-c: is-cat-limit
```

```
    α ‹c ↓CF 𝔎› 𝔄 ‹𝔗 ∘CF c ○CF 𝔎› ‹?UObj c› ‹?UArr c›
    by (rule assms(3)[OF that(3)])
  from that have b: b ∈∘ 𝔅(|Obj|) and f: f : c ↦𝔈 𝔎(|ObjMap|)(|b|)
    by blast+
  show
    ε(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|) =
      ?UArr c(|NTMap|)(|a, b, f|)• ∘A𝔄 σ(|NTMap|)(|c|)
    using ε′-NTMap-app-def[OF that(2,3)] that(2,3)
    unfolding that(1)
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-Kan-cs-intros
      )
qed

show ∃!σ.
  σ : 𝔊 ↦CF ?the-cf-rKe : 𝔈 ↦↦Cα 𝔄 ∧
  ε = ?the-ntcf-rKe •NTCF (σ ∘NTCF-CF 𝔎)
proof(intro ex1I[where a=σ] conjI; (elim conjE)?)

  define τ where τ a b f =
    [
      (
        λF∈∘b ↓CF 𝔎(|Obj|).
          ?UArr b(|NTMap|)(|F|) ∘A𝔄 σ(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|)
      ),
      cf-const (b ↓CF 𝔎) 𝔄 (𝔊(|ObjMap|)(|a|)),
      𝔗 ∘CF b ○CF 𝔎,
      b ↓CF 𝔎,
      𝔄
    ]∘
    for a b f

  have τ-components:
    τ a b f(|NTMap|) =
      (
        λF∈∘b ↓CF 𝔎(|Obj|).
          ?UArr b(|NTMap|)(|F|) ∘A𝔄 σ(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|)
      )
    τ a b f(|NTDom|) = cf-const (b ↓CF 𝔎) 𝔄 (𝔊(|ObjMap|)(|a|))
    τ a b f(|NTCod|) = 𝔗 ∘CF b ○CF 𝔎
    τ a b f(|NTDGDom|) = b ↓CF 𝔎
    τ a b f(|NTDGCod|) = 𝔄
    for a b f
    unfolding τ-def nt-field-simps by (simp-all add: nat-omega-simps)
  note [cat-Kan-cs-simps] = τ-components(2–5)
  have τ-NTMap-app[cat-Kan-cs-simps]:
    τ a b f(|NTMap|)(|F|) =
      ?UArr b(|NTMap|)(|F|) ∘A𝔄 σ(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|)
    if F ∈∘ b ↓CF 𝔎(|Obj|) for a b f F
    using that unfolding τ-components by auto

  have τ: τ a b f :
    𝔊(|ObjMap|)(|a|) <CF.cone 𝔗 ∘CF b ○CF 𝔎 : b ↓CF 𝔎 ↦↦Cα 𝔄
    if f-is-arr: f : a ↦𝔈 b for a b f
  proof–
```

note f = $\mathfrak{K}$.*HomCod.cat-is-arrD*[*OF that*]
note *lim-a* = *assms*(*3*)[*OF f*(*2*)] **and** *lim-b* = *assms*(*3*)[*OF f*(*3*)]

interpret *lim-b*: *is-cat-limit*
α ‹b $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ b $_O\sqcap_{CF}$ $\mathfrak{K}$› ‹*?UObj b*› ‹*?UArr b*›
by (*rule lim-b*)

note *lim-b.cat-cone-Comp-commute*[*cat-cs-simps del*]

from f **have** a: $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** b: $b \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*

show *?thesis*
proof(*intro is-cat-coneI is-ntcfI'*)

show *vfsequence* (τ a b f) **unfolding** τ-*def* **by** *simp*
show *vcard* (τ a b f) = $5_{\mathbb{N}}$
unfolding τ-*def* **by** (*simp add*: *nat-omega-simps*)
show *vsv* (τ a b $f(\!|NTMap|\!)$) **unfolding** τ-*components* **by** *auto*
show $\mathcal{D}_\circ$ (τ a b $f(\!|NTMap|\!)$) = b $\downarrow_{CF}$ $\mathfrak{K}(\!|Obj|\!)$ **by** (*auto simp*: τ-*components*)
show τ a b $f(\!|NTMap|\!)(\!|A|\!)$:
cf-const (b $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$ ($\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$)($|ObjMap|\!)(\!|A|\!)$ $\mapsto_{\mathfrak{A}}$
($\mathfrak{T}$ $\circ_{CF}$ b $_O\sqcap_{CF}$ $\mathfrak{K}$)($|ObjMap|\!)(\!|A|\!)$
if $A \in_\circ b \downarrow_{CF} \mathfrak{K}(\!|Obj|\!)$ **for** A
proof–
from *that f-is-arr* **obtain** b' f'
where *A-def*: $A = [0, b', f']_\circ$
and b': $b' \in_\circ \mathfrak{B}(\!|Obj|\!)$
and f': $f' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\!|ObjMap|\!)(\!|b'|\!)$
by *auto*
from *f-is-arr that* b' f' a b **show** *?thesis*
unfolding *A-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros*
)
qed
show
τ a b $f(\!|NTMap|\!)(\!|B|\!)$ $\circ_{A\mathfrak{A}}$
cf-const (b $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$ ($\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$)($|ArrMap|\!)(\!|F|\!)$ =
($\mathfrak{T}$ $\circ_{CF}$ b $_O\sqcap_{CF}$ $\mathfrak{K}$)($|ArrMap|\!)(\!|F|\!)$ $\circ_{A\mathfrak{A}}$ τ a b $f(\!|NTMap|\!)(\!|A|\!)$
if $F : A \mapsto_{b \downarrow_{CF} \mathfrak{K}} B$ **for** A B F
proof–
from *that* **have** *F*: $F : A \mapsto_{b \downarrow_{CF} \mathfrak{K}} B$
by (*auto intro*: *is-arrI*)
with *f-is-arr* **obtain** b' f' b'' f'' h'
where *F-def*: $F = [[0, b', f']_\circ, [0, b'', f'']_\circ, [0, h']_\circ]_\circ$
and *A-def*: $A = [0, b', f']_\circ$
and *B-def*: $B = [0, b'', f'']_\circ$
and h': $h' : b' \mapsto_{\mathfrak{B}} b''$
and f': $f' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\!|ObjMap|\!)(\!|b'|\!)$
and f'': $f'' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\!|ObjMap|\!)(\!|b''|\!)$
and f''-*def*: $\mathfrak{K}(\!|ArrMap|\!)(\!|h'|\!)$ $\circ_{A\mathfrak{C}}$ $f' = f''$
by *auto*
from
lim-b.ntcf-Comp-commute[*OF that*]

```
      that f-is-arr g′ h′ f′ f″
    have [cat-Kan-cs-simps]:
      ?UArr b(|NTMap|)(|0, b″, ℜ(|ArrMap|)(|h′|) ∘_{A ℭ} f′|)• =
        𝔗(|ArrMap|)(|h′|) ∘_{A 𝔄} ?UArr b(|NTMap|)(|0, b′, f′|)•
      unfolding F-def A-def B-def
      by
        (
          cs-prems
            cs-simp:
              cat-cs-simps cat-comma-cs-simps f″-def[symmetric]
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
    from f-is-arr that g′ h′ f′ f″ show ?thesis
      unfolding F-def A-def B-def
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-Kan-cs-simps
              cat-comma-cs-simps
              f″-def[symmetric]
            cs-intro:
              cat-cs-intros cat-Kan-cs-intros cat-comma-cs-intros
        )+
  qed

qed
  (
    use that f-is-arr in
      ‹
        cs-concl
          cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
      ›
  )+
qed

show σ: σ : 𝔊 ↦_{CF} ?the-cf-rKe : ℭ ↦↦_{Cα} 𝔄
proof(rule is-ntcfI′)

  show vfsequence σ unfolding σ-def by simp
  show vcard σ = 5_ℕ unfolding σ-def by (simp add: nat-omega-simps)
  show vsv (σ(|NTMap|)) unfolding σ-components by auto
  show 𝒟_∘ (σ(|NTMap|)) = ℭ(|Obj|) unfolding σ-components by simp
  show σ(|NTMap|)(|a|) : 𝔊(|ObjMap|)(|a|) ↦_𝔄 ?the-cf-rKe(|ObjMap|)(|a|)
    if a ∈_∘ ℭ(|Obj|) for a
    using that
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-Kan-cs-intros
      )

  then have [cat-Kan-cs-intros]: σ(|NTMap|)(|a|) : b ↦_𝔄 c
    if a ∈_∘ ℭ(|Obj|)
      and b = 𝔊(|ObjMap|)(|a|)
      and c = ?the-cf-rKe(|ObjMap|)(|a|)
```

```
  for a b c
  using that(1) unfolding that(2,3) by simp

show
  σ(|NTMap|)(|b|) ∘_{A𝔄} 𝔊(|ArrMap|)(|f|) =
    ?the-cf-rKe(|ArrMap|)(|f|) ∘_{A𝔄} σ(|NTMap|)(|a|)
  if f-is-arr: f : a ↦_ℭ b for a b f
proof–

  note f = ℜ.HomCod.cat-is-arrD[OF that]
  note lim-a = assms(3)[OF f(2)] and lim-b = assms(3)[OF f(3)]

  interpret lim-a: is-cat-limit
    α ‹a ↓_{CF} ℜ› 𝔄 ‹𝔗 ∘_{CF} a _O⨅_{CF} ℜ› ‹?UObj a› ‹?UArr a›
    by (rule lim-a)
  interpret lim-b: is-cat-limit
    α ‹b ↓_{CF} ℜ› 𝔄 ‹𝔗 ∘_{CF} b _O⨅_{CF} ℜ› ‹?UObj b› ‹?UArr b›
    by (rule lim-b)

  from f have a: a ∈_∘ ℭ(|Obj|) and b: b ∈_∘ ℭ(|Obj|) by auto

  from lim-b.cat-lim-unique-cone'[OF τ[OF that]] obtain g'
    where g': g' : 𝔊(|ObjMap|)(|a|) ↦_𝔄 ?UObj b
      and τ-NTMap-app: ⋀A. A ∈_∘ (b ↓_{CF} ℜ(|Obj|)) ⟹
        τ a b f(|NTMap|)(|A|) = ?UArr b(|NTMap|)(|A|) ∘_{A𝔄} g'
      and g'-unique: ⋀g''.
        [[
          g'' : 𝔊(|ObjMap|)(|a|) ↦_𝔄 ?UObj b;
          ⋀A. A ∈_∘ b ↓_{CF} ℜ(|Obj|) ⟹
            τ a b f(|NTMap|)(|A|) = ?UArr b(|NTMap|)(|A|) ∘_{A𝔄} g''
        ]] ⟹ g'' = g'
    by metis

  have lim-Obj-a-fℜ[symmetric, cat-Kan-cs-simps]:
    ?UArr a(|NTMap|)(|a', b', f'|) ∘_{Aℭ} f(|)• =
      ?UArr b(|NTMap|)(|A|) ∘_{A𝔄} ?the-cf-rKe(|ArrMap|)(|f|)
    if A = [a', b', f']_∘ and A ∈_∘ b ↓_{CF} ℜ(|Obj|) for A a' b' f'
  proof–
    from that(2) f-is-arr have a'-def: a' = 0
      and b': b' ∈_∘ 𝔅(|Obj|)
      and f': f' : b ↦_ℭ ℜ(|ObjMap|)(|b'|)
      unfolding that(1) by auto
    show ?thesis
      unfolding that(1)
      by
        (
          rule
            lim-Obj-the-cf-rKe-commute
              [
                where lim-Obj=lim-Obj,
                OF
                  assms(1,2)
                  lim-a
                  lim-b
                  f-is-arr
                  that(2)[unfolded that(1)]
              ]
        )
```

```
qed
{
  fix a′ b′ f′ A
  note 𝔗.HomCod.cat-assoc-helper[
    where h=‹?UArr b(|NTMap|)(|a′,b′,f′|)•›
      and g=‹?the-cf-rKe(|ArrMap|)(|f|)›
      and q=‹?UArr a(|NTMap|)(|a′, b′, f′ ∘A𝔈 f|)•›
      ]
}
note [cat-Kan-cs-simps] = this

show ?thesis
proof(rule trans-sym[where s=g′])
  show σ(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|) = g′
  proof(rule g′-unique)
    from that show
      σ(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|) : 𝔊(|ObjMap|)(|a|) ↦𝔄 ?UObj b
      by (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros)
    fix A assume prems′: A ∈∘ b ↓CF 𝔎(|Obj|)
    with f-is-arr obtain b′ f′
      where A-def: A = [0, b′, f′]∘
        and b′: b′ ∈∘ 𝔅(|Obj|)
        and f′: f′ : b ↦𝔈 𝔎(|ObjMap|)(|b′|)
      by auto
    from f-is-arr prems′ show
      τ a b f(|NTMap|)(|A|) =
        ?UArr b(|NTMap|)(|A|) ∘A𝔄 (σ(|NTMap|)(|b|) ∘A𝔄 𝔊(|ArrMap|)(|f|))
      unfolding A-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-Kan-cs-simps
            cs-intro: cat-cs-intros cat-Kan-cs-intros
        )
  qed
  show ?the-cf-rKe(|ArrMap|)(|f|) ∘A𝔄 σ(|NTMap|)(|a|) = g′
  proof(rule g′-unique)
    fix A assume prems′: A ∈∘ b ↓CF 𝔎(|Obj|)
    with f-is-arr obtain b′ f′
      where A-def: A = [0, b′, f′]∘
        and b′: b′ ∈∘ 𝔅(|Obj|)
        and f′: f′ : b ↦𝔈 𝔎(|ObjMap|)(|b′|)
      by auto
    {
      fix a′ b′ f′ A
      note 𝔗.HomCod.cat-assoc-helper
        [
          where h=‹?UArr b(|NTMap|)(|a′, b′, f′|)•›
            and g=‹σ(|NTMap|)(|b|)›
            and q=‹ε(|NTMap|)(|b′|) ∘A𝔄 𝔊(|ArrMap|)(|f′|)›
        ]
    }
    note [cat-Kan-cs-simps] =
      this
      εb-𝔊f[OF A-def prems′ b, symmetric]
      εb-𝔊f[symmetric]
    from f-is-arr prems′ b′ f′ show
      τ a b f(|NTMap|)(|A|) =
```

```
        ?UArr b(|NTMap|)(|A|) ∘_A𝔄
          (?the-cf-rKe(|ArrMap|)(|f|) ∘_A𝔄 σ(|NTMap|)(|a|))
      unfolding A-def
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-Kan-cs-simps
              cat-comma-cs-simps
              cat-op-simps
            cs-intro:
              cat-cs-intros
              cat-Kan-cs-intros
              cat-comma-cs-intros
              cat-op-intros
        )
    qed
      (
        use that in
          ‹
            cs-concl
              cs-simp: cat-Kan-cs-simps
              cs-intro: cat-cs-intros cat-Kan-cs-intros
          ›
      )
  qed
 qed
 (
   cs-concl cs-shallow
     cs-simp: cat-cs-simps cat-Kan-cs-simps
     cs-intro: cat-cs-intros
 )+
then interpret σ: is-ntcf α ℭ 𝔄 𝔊 ‹?the-cf-rKe› σ by simp

show ε = ?the-ntcf-rKe ⊙_NTCF (σ ∘_NTCF-CF ℜ)
proof(rule ntcf-eqI)
  have dom-lhs: 𝒟∘ (ε(|NTMap|)) = 𝔅(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have dom-rhs: 𝒟∘ ((?the-ntcf-rKe ⊙_NTCF (σ ∘_NTCF-CF ℜ))(|NTMap|)) = 𝔅(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show ε(|NTMap|) = (?the-ntcf-rKe ⊙_NTCF (σ ∘_NTCF-CF ℜ))(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix b assume prems': b ∈∘ 𝔅(|Obj|)
    note [cat-Kan-cs-simps] = ε b-𝔊f[
      where f=‹ℭ(|CId|)(|ℜ(|ObjMap|)(|b|)|)› and c=‹ℜ(|ObjMap|)(|b|)›, symmetric
      ]
    from prems' σ show
      ε(|NTMap|)(|b|) = (?the-ntcf-rKe ⊙_NTCF (σ ∘_NTCF-CF ℜ))(|NTMap|)(|b|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
```

```
fix σ′ assume prems′:
  σ′ : 𝔊 ↦_CF ?the-cf-rKe : ℭ ↦↦_Cα 𝔄
  ε = ?the-ntcf-rKe ⋆_NTCF (σ′ ∘_NTCF−CF ℜ)

interpret σ′: is-ntcf α ℭ 𝔄 𝔊 ‹?the-cf-rKe› σ′ by (rule prems′(1))

have ε-NTMap-app[symmetric, cat-Kan-cs-simps]:
  ε(|NTMap|)(|b′|) =
    ?UArr (ℜ(|ObjMap|)(|b′|))(|NTMap|)(|a′, b′, ℭ(|CId|)(|ℜ(|ObjMap|)(|b′|)|)|)• ∘_A 𝔄
    σ′(|NTMap|)(|ℜ(|ObjMap|)(|b′|)|)
  if b′ ∈_∘ 𝔅(|Obj|) and a′ = 0 for a′ b′
proof–
  from prems′(2) have ε-NTMap-app:
    ε(|NTMap|)(|b′|) = (?the-ntcf-rKe ⋆_NTCF (σ′ ∘_NTCF−CF ℜ))(|NTMap|)(|b′|)
    for b′
    by simp
  show ?thesis
    using ε-NTMap-app[of b′] that(1)
    unfolding that(2)
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed
{
  fix a′ b′ f′ A
  note 𝔗.HomCod.cat-assoc-helper
    [
      where h= ‹?UArr (ℜ(|ObjMap|)(|b′|))(|NTMap|)(|a′, b′, ℭ(|CId|)(|ℜ(|ObjMap|)(|b′|)|)|)•›
        and g=‹σ′(|NTMap|)(|ℜ(|ObjMap|)(|b′|)|)›
        and q=‹ε(|NTMap|)(|b′|)›
    ]
}
note [cat-Kan-cs-simps] = this εb-𝔊f[symmetric]
{
  fix a′ b′ f′ A
  note 𝔗.HomCod.cat-assoc-helper
    [
      where h=
        ‹?UArr (ℜ(|ObjMap|)(|b′|))(|NTMap|)(|a′, b′, ℭ(|CId|)(|ℜ(|ObjMap|)(|b′|)|)|)•›
      and g=‹σ(|NTMap|)(|ℜ(|ObjMap|)(|b′|)|)›
      and q=‹ε(|NTMap|)(|b′|)›
    ]
}
note [cat-Kan-cs-simps] = this

show σ′ = σ
proof(rule ntcf-eqI)

  show σ′ : 𝔊 ↦_CF ?the-cf-rKe : ℭ ↦↦_Cα 𝔄 by (rule prems′(1))
  show σ : 𝔊 ↦_CF ?the-cf-rKe : ℭ ↦↦_Cα 𝔄 by (rule σ)

  have dom-lhs: 𝒟_∘ (σ(|NTMap|)) = ℭ(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have dom-rhs: 𝒟_∘ (σ′(|NTMap|)) = ℭ(|Obj|)
```

```
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)

show σ'(|NTMap|) = σ(|NTMap|)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)

  fix c assume prems': c ∈∘ ℭ(|Obj|)

  note lim-c = assms(3)[OF prems']
  interpret lim-c: is-cat-limit
    α ‹c ↓CF ℜ› 𝔄 ‹𝔗 ∘CF c ∘CF ⨅CF ℜ› ‹?UObj c› ‹?UArr c›
    by (rule lim-c)
  from prems' have CId-c: ℭ(|CId|)(|c|) : c ↦ℭ c
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)

  from lim-c.cat-lim-unique-cone'[OF τ[OF CId-c]] obtain f
    where f: f : 𝔊(|ObjMap|)(|c|) ↦𝔄 ?UObj c
      and ⋀A. A ∈∘ c ↓CF ℜ(|Obj|) ⟹
        τ c c (ℭ(|CId|)(|c|))(|NTMap|)(|A|) = ?UArr c(|NTMap|)(|A|) ∘A𝔄 f
      and f-unique: ⋀f'.
        [[
          f' : 𝔊(|ObjMap|)(|c|) ↦𝔄 ?UObj c;
          ⋀A. A ∈∘ c ↓CF ℜ(|Obj|) ⟹
            τ c c (ℭ(|CId|)(|c|))(|NTMap|)(|A|) = ?UArr c(|NTMap|)(|A|) ∘A𝔄 f'
        ]] ⟹ f' = f
    by metis

  note [symmetric, cat-cs-simps] =
    σ.ntcf-Comp-commute
    σ'.ntcf-Comp-commute

  show σ'(|NTMap|)(|c|) = σ(|NTMap|)(|c|)
  proof(rule trans-sym[where s=f])

    show σ'(|NTMap|)(|c|) = f
    proof(rule f-unique)

      fix A assume prems'': A ∈∘ c ↓CF ℜ(|Obj|)

      with prems' obtain b' f'
        where A-def: A = [0, b', f']∘
          and b': b' ∈∘ 𝔅(|Obj|)
          and f': f' : c ↦ℭ ℜ(|ObjMap|)(|b'|)
        by auto

      let ?ℜb' = ‹ℜ(|ObjMap|)(|b'|)›

      from b' have ℜb': ?ℜb' ∈∘ ℭ(|Obj|)
        by (cs-concl cs-shallow cs-intro: cat-cs-intros)

      interpret lim-ℜb': is-cat-limit
        α ‹?ℜb' ↓CF ℜ› 𝔄 ‹𝔗 ∘CF ?ℜb' ∘CF ⨅CF ℜ› ‹?UObj ?ℜb'› ‹?UArr ?ℜb'›
        by (rule assms(3)[OF ℜb'])

      from ℜb' have CId-ℜb': ℭ(|CId|)(|?ℜb'|) : ?ℜb' ↦ℭ ?ℜb'
        by (cs-concl cs-intro: cat-cs-intros)
      from CId-ℜb' b' have a'-b'-CId-ℜb':
        [0, b', ℭ(|CId|)(|?ℜb'|)]∘ ∈∘ ?ℜb' ↓CF ℜ(|Obj|)
        by
```

```
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )
  from
    lim-Obj-the-cf-rKe-commute[
      where lim-Obj=lim-Obj,
      OF assms(1,2) lim-c assms(3)[OF ℜb'] f' a'-b'-CId-ℜb'
    ]
    f'
  have [cat-Kan-cs-simps]:
    ?UArr c(|NTMap|)(|0, b', f'|)• =
      ?UArr ?ℜb'(|NTMap|)(|0, b', ℭ(|CId|)(|?ℜb'|)|)• ∘A𝔄
        ?the-cf-rKe(|ArrMap|)(|f'|)
    by (cs-prems cs-shallow cs-simp: cat-cs-simps)

  from prems' prems'' b' f' show
    τ c c (ℭ(|CId|)(|c|))(|NTMap|)(|A|) = ?UArr c(|NTMap|)(|A|) ∘A𝔄 σ'(|NTMap|)(|c|)
    unfolding A-def
    by
      (
        cs-concl
          cs-simp:
            cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
          cs-intro:
            cat-lim-cs-intros
            cat-cs-intros
            cat-comma-cs-intros
            cat-Kan-cs-intros
      )
qed
  (
    use prems' in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
      ›
  )

show σ(|NTMap|)(|c|) = f
proof(rule f-unique)
  fix A assume prems'': A ∈∘ c ↓CF ℜ(|Obj|)
  from this prems' obtain b' f'
    where A-def: A = [0, b', f']∘
      and b': b' ∈∘ 𝔅(|Obj|)
      and f': f' : c ↦ℭ ℜ(|ObjMap|)(|b'|)
    by auto
  let ?ℜb' = ‹ℜ(|ObjMap|)(|b'|)›
  from b' have ℜb': ?ℜb' ∈∘ ℭ(|Obj|)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  interpret lim-ℜb': is-cat-limit
    α ‹?ℜb' ↓CF ℜ› 𝔄 ‹𝔗 ∘CF ?ℜb' ∘CF ℜ› ‹?UObj ?ℜb'› ‹?UArr ?ℜb'›
    by (rule assms(3)[OF ℜb'])
  from ℜb' have CId-ℜb': ℭ(|CId|)(|?ℜb'|) : ?ℜb' ↦ℭ ?ℜb'
    by (cs-concl cs-intro: cat-cs-intros)
  from CId-ℜb' b' have a'-b'-CId-ℜb':
    [0, b', ℭ(|CId|)(|?ℜb'|)]∘ ∈∘ ?ℜb' ↓CF ℜ(|Obj|)
```

```
            by
              (
                cs-concl cs-shallow
                  cs-simp: cat-cs-simps cat-comma-cs-simps
                  cs-intro: cat-cs-intros cat-comma-cs-intros
              )

          from
            lim-Obj-the-cf-rKe-commute
              [
                where lim-Obj=lim-Obj,
                OF assms(1,2) lim-c assms(3)[OF ℜb'] f' a'-b'-CId-ℜb'
              ]
            f'
          have [cat-Kan-cs-simps]:
            ?UArr c⦇NTMap⦈⦇0, b', f'⦈• =
              ?UArr (?ℜb')⦇NTMap⦈⦇0, b', ℭ⦇CId⦈⦇?ℜb'⦈⦈• ∘A𝔄
                ?the-cf-rKe⦇ArrMap⦈⦇f'⦈
            by (cs-prems cs-shallow cs-simp: cat-cs-simps)
          from prems' prems'' b' f' show
            τ c c (ℭ⦇CId⦈⦇c⦈)⦇NTMap⦈⦇A⦈ = ?UArr c⦇NTMap⦈⦇A⦈ ∘A𝔄 σ⦇NTMap⦈⦇c⦈
            unfolding A-def
            by
              (
                cs-concl
                  cs-simp:
                    cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
                  cs-intro:
                    cat-lim-cs-intros
                    cat-cs-intros
                    cat-comma-cs-intros
                    cat-Kan-cs-intros
              )
        qed
          (
            use prems' in
              ‹
                cs-concl cs-shallow
                  cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
              ›
          )
      qed

    qed auto

  qed simp-all

 qed

qed (cs-concl cs-shallow cs-intro: cat-cs-intros)+

qed

lemma the-ntcf-lKe-is-cat-lKe:
  assumes ℜ : 𝔅 ↦↦Cα ℭ
    and 𝔗 : 𝔅 ↦↦Cα 𝔄
    and ⋀c. c ∈∘ ℭ⦇Obj⦈ ⟹ lim-Obj c⦇UArr⦈ :
      𝔗 ∘CF ℜ CF⨅O c >CF.colim lim-Obj c⦇UObj⦈ : ℜ CF↓ c ↦↦Cα 𝔄
```

```
  shows the-ntcf-lKe α 𝔗 𝔎 lim-Obj :
    𝔗 ↦_{CF.lKeα} the-cf-lKe α 𝔗 𝔎 lim-Obj ∘_{CF} 𝔎 : 𝔅 ↦_C ℭ ↦_C 𝔄
proof–
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  {
    fix c assume prems: c ∈_∘ ℭ⟦Obj⟧
    from assms(3)[OF this] have lim-Obj-UArr: lim-Obj c⟦UArr⟧ :
      𝔗 ∘_{CF} 𝔎 _{CF}↑_O c >_{CF.colim} lim-Obj c⟦UObj⟧ : 𝔎 _{CF}↓ c ↦↦_{Cα} 𝔄.
    then interpret lim-Obj-c: is-cat-colimit
      α ‹𝔎 _{CF}↓ c› 𝔄 ‹𝔗 ∘_{CF} 𝔎 _{CF}↑_O c› ‹lim-Obj c⟦UObj⟧› ‹lim-Obj c⟦UArr⟧›
      by simp
    note op-ua-UArr-is-cat-limit'[
      where lim-Obj=lim-Obj, OF assms(1,2) prems lim-Obj-UArr
      ]
  }
  note the-ntcf-rKe-is-cat-rKe = the-ntcf-rKe-is-cat-rKe
    [
      OF 𝔎.is-functor-op 𝔗.is-functor-op,
      unfolded cat-op-simps,
      where lim-Obj=‹op-ua lim-Obj 𝔎›,
      unfolded cat-op-simps,
      OF this,
      simplified,
      folded the-cf-lKe-def the-ntcf-lKe-def
    ]
  show ?thesis
    by
      (
        rule is-cat-rKe.is-cat-lKe-op
          [
            OF the-ntcf-rKe-is-cat-rKe,
            unfolded cat-op-simps,
            folded the-cf-lKe-def the-ntcf-lKe-def
          ]
      )
qed
```

14.5 Preservation of Kan extensions

The following definitions are similar to the definitions that can be found in [14] or [8].

```
locale is-cat-rKe-preserves =
  is-cat-rKe α 𝔅 ℭ 𝔄 𝔎 𝔗 𝔊 ε + is-functor α 𝔄 𝔇 ℌ
  for α 𝔅 ℭ 𝔄 𝔇 𝔎 𝔗 𝔊 ℌ ε +
  assumes cat-rKe-preserves:
    ℌ ∘_{CF-NTCF} ε : (ℌ ∘_{CF} 𝔊) ∘_{CF} 𝔎 ↦_{CF.rKeα} ℌ ∘_{CF} 𝔗 : 𝔅 ↦_C ℭ ↦_C 𝔇

syntax -is-cat-rKe-preserves ::
  V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ bool
  (
    ‹(- :/ - ∘_{CF} - ↦_{CF.rKe1} - :/ - ↦_C - ↦_C - : - ↦↦_C -)›
    [51, 51, 51, 51, 51, 51, 51, 51, 51] 51
  )
syntax-consts -is-cat-rKe-preserves ⇌ is-cat-rKe-preserves
translations ε : 𝔊 ∘_{CF} 𝔎 ↦_{CF.rKeα} 𝔗 : 𝔅 ↦_C ℭ ↦_C (ℌ : 𝔄 ↦↦_C 𝔇) ⇌
  CONST is-cat-rKe-preserves α 𝔅 ℭ 𝔄 𝔇 𝔎 𝔗 𝔊 ℌ ε
```

locale *is-cat-lKe-preserves* =
  is-cat-lKe α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ η + *is-functor* α $\mathfrak{A}$ $\mathfrak{D}$ $\mathfrak{H}$
  for α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{D}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ $\mathfrak{H}$ η +
  assumes *cat-lKe-preserves*:
    $\mathfrak{H} \circ_{CF-NTCF} \eta : \mathfrak{H} \circ_{CF} \mathfrak{T} \mapsto_{CF.lKe\alpha} (\mathfrak{H} \circ_{CF} \mathfrak{F}) \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{D}$

syntax *-is-cat-lKe-preserves* ::
  $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (
    ‹(- :/ - $\mapsto_{CF.lKe1}$ - $\circ_{CF}$ - :/ - $\mapsto_C$ - $\mapsto_C$ - : - $\mapsto\mapsto_C$ -)›
    [*51*, *51*, *51*, *51*, *51*, *51*, *51*, *51*, *51*] *51*
)
syntax-consts *-is-cat-lKe-preserves* $\rightleftharpoons$ *is-cat-lKe-preserves*
translations $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_C \mathfrak{D}) \rightleftharpoons$
  CONST is-cat-lKe-preserves α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{D}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ $\mathfrak{H}$ η

Rules.

lemma (**in** *is-cat-rKe-preserves*) *is-cat-rKe-preserves-axioms'*:
  assumes $\alpha' = \alpha$
    and $\mathfrak{G}' = \mathfrak{G}$
    and $\mathfrak{K}' = \mathfrak{K}$
    and $\mathfrak{T}' = \mathfrak{T}$
    and $\mathfrak{H}' = \mathfrak{H}$
    and $\mathfrak{B}' = \mathfrak{B}$
    and $\mathfrak{A}' = \mathfrak{A}$
    and $\mathfrak{C}' = \mathfrak{C}$
    and $\mathfrak{D}' = \mathfrak{D}$
  shows $\varepsilon : \mathfrak{G}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe_{\alpha'}} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto\mapsto_C \mathfrak{D}')$
  unfolding *assms* **by** (*rule is-cat-rKe-preserves-axioms*)

mk-ide rf *is-cat-rKe-preserves-def*[*unfolded is-cat-rKe-preserves-axioms-def*]
  |*intro is-cat-rKe-preservesI*|
  |*dest is-cat-rKe-preservesD*[*dest*]|
  |*elim is-cat-rKe-preservesE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-rKeD*(*1–3*)

lemma (**in** *is-cat-lKe-preserves*) *is-cat-lKe-preserves-axioms'*:
  assumes $\alpha' = \alpha$
    and $\mathfrak{F}' = \mathfrak{F}$
    and $\mathfrak{K}' = \mathfrak{K}$
    and $\mathfrak{T}' = \mathfrak{T}$
    and $\mathfrak{H}' = \mathfrak{H}$
    and $\mathfrak{B}' = \mathfrak{B}$
    and $\mathfrak{A}' = \mathfrak{A}$
    and $\mathfrak{C}' = \mathfrak{C}$
    and $\mathfrak{D}' = \mathfrak{D}$
  shows $\eta : \mathfrak{T}' \mapsto_{CF.lKe\alpha} \mathfrak{F}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto\mapsto_C \mathfrak{D}')$
  unfolding *assms* **by** (*rule is-cat-lKe-preserves-axioms*)

mk-ide rf *is-cat-lKe-preserves-def*[*unfolded is-cat-lKe-preserves-axioms-def*]
  |*intro is-cat-lKe-preservesI*|
  |*dest is-cat-lKe-preservesD*[*dest*]|
  |*elim is-cat-lKe-preservesE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-lKe-preservesD*(*1–3*)

Duality.

lemma (**in** *is-cat-rKe-preserves*) *is-cat-rKe-preserves-op*:
  op-ntcf ε :
    op-cf $\mathfrak{T} \mapsto_{CF.lKe\alpha}$ *op-cf* $\mathfrak{G} \circ_{CF}$ *op-cf* $\mathfrak{K}$:
    op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C}$ (*op-cf* $\mathfrak{H}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C}$ *op-cat* $\mathfrak{D}$)
proof(*intro is-cat-lKe-preservesI*)
  from *cat-rKe-preserves* **show** *op-cf* $\mathfrak{H} \circ_{CF-NTCF}$ *op-ntcf* ε :
    op-cf $\mathfrak{H} \circ_{CF}$ *op-cf* $\mathfrak{T} \mapsto_{CF.lKe\alpha}$ (*op-cf* $\mathfrak{H} \circ_{CF}$ *op-cf* $\mathfrak{G}$) $\circ_{CF}$ *op-cf* $\mathfrak{K}$:
    op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C}$ *op-cat* $\mathfrak{D}$
    by (*cs-concl-step op-ntcf-cf-ntcf-comp*[*symmetric*])
      (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)
qed (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)+

lemma (**in** *is-cat-rKe-preserves*) *is-cat-lKe-preserves-op'*[*cat-op-intros*]:
  assumes $\mathfrak{T}'$ = *op-cf* $\mathfrak{T}$
    and $\mathfrak{G}'$ = *op-cf* $\mathfrak{G}$
    and $\mathfrak{K}'$ = *op-cf* $\mathfrak{K}$
    and $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
    and $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$
    and $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
    and $\mathfrak{D}'$ = *op-cat* $\mathfrak{D}$
    and $\mathfrak{H}'$ = *op-cf* $\mathfrak{H}$
  shows *op-ntcf* ε :
    $\mathfrak{T}' \mapsto_{CF.lKe\alpha}$ $\mathfrak{G}' \circ_{CF}$ $\mathfrak{K}' : \mathfrak{B}' \mapsto_{C} \mathfrak{C}' \mapsto_{C} (\mathfrak{H}' : \mathfrak{A}' \mapsto\mapsto_{C} \mathfrak{D}')$
  unfolding *assms* **by** (*rule is-cat-rKe-preserves-op*)

lemmas [*cat-op-intros*] = *is-cat-rKe-preserves.is-cat-lKe-preserves-op'*

lemma (**in** *is-cat-lKe-preserves*) *is-cat-rKe-preserves-op*:
  op-ntcf η :
    op-cf $\mathfrak{F} \circ_{CF}$ *op-cf* $\mathfrak{K} \mapsto_{CF.rKe\alpha}$ *op-cf* $\mathfrak{T}$:
    op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C}$ (*op-cf* $\mathfrak{H}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C}$ *op-cat* $\mathfrak{D}$)
proof(*intro is-cat-rKe-preservesI*)
  from *cat-lKe-preserves* **show** *op-cf* $\mathfrak{H} \circ_{CF-NTCF}$ *op-ntcf* η :
    (*op-cf* $\mathfrak{H} \circ_{CF}$ *op-cf* $\mathfrak{F}$) $\circ_{CF}$ *op-cf* $\mathfrak{K} \mapsto_{CF.rKe\alpha}$ *op-cf* $\mathfrak{H} \circ_{CF}$ *op-cf* $\mathfrak{T}$:
    op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C}$ *op-cat* $\mathfrak{D}$
    by (*cs-concl-step op-ntcf-cf-ntcf-comp*[*symmetric*])
      (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)
qed (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-op-intros*)+

lemma (**in** *is-cat-lKe-preserves*) *is-cat-rKe-preserves-op'*[*cat-op-intros*]:
  assumes $\mathfrak{T}'$ = *op-cf* $\mathfrak{T}$
    and $\mathfrak{F}'$ = *op-cf* $\mathfrak{F}$
    and $\mathfrak{K}'$ = *op-cf* $\mathfrak{K}$
    and $\mathfrak{H}'$ = *op-cf* $\mathfrak{H}$
    and $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
    and $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$
    and $\mathfrak{C}'$ = *op-cat* $\mathfrak{C}$
    and $\mathfrak{D}'$ = *op-cat* $\mathfrak{D}$
  shows *op-ntcf* η :
    $\mathfrak{F}' \circ_{CF}$ $\mathfrak{K}' \mapsto_{CF.rKe\alpha}$ $\mathfrak{T}' : \mathfrak{B}' \mapsto_{C} \mathfrak{C}' \mapsto_{C} (\mathfrak{H}' : \mathfrak{A}' \mapsto\mapsto_{C} \mathfrak{D}')$
  unfolding *assms* **by** (*rule is-cat-rKe-preserves-op*)

14.6 All concepts are Kan extensions

Background information for this subsection is provided in Chapter X-7 in [9] and subsection 6.5 in [14]. It should be noted that only the connections between the Kan extensions, limits and adjunctions are exposed (an alternative proof of the Yoneda lemma using Kan extensions is not

provided in the context of this work).

14.6.1 Limits and colimits

lemma *cat-rKe-is-cat-limit*:
— The statement of the theorem is similar to the statement of a part of Theorem 1 in Chapter X-7 in [9] or Proposition 6.5.1 in [14].
assumes $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_{C} \mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f} \mapsto_{C} \mathfrak{A}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
shows $\varepsilon : \mathfrak{G}(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{a}|\!) <_{CF.lim} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–

interpret ε: *is-cat-rKe* $\alpha\ \mathfrak{B}\ \langle \mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f}\rangle\ \mathfrak{A}\ \mathfrak{K}\ \mathfrak{T}\ \mathfrak{G}\ \varepsilon$ **by** (*rule assms*(*1*))
interpret $\mathfrak{T}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{T}$ **by** (*rule assms*(*2*))

from *cat-1-components*(*1*) **have** $\mathfrak{a}$: $\mathfrak{a} \in_{\circ} \mathit{Vset}\ \alpha$
by (*auto simp*: ε.*AG.HomCod.cat-in-Obj-in-Vset*)
from *cat-1-components*(*2*) **have** $\mathfrak{f}$: $\mathfrak{f} \in_{\circ} \mathit{Vset}\ \alpha$
by (*auto simp*: ε.*AG.HomCod.cat-in-Arr-in-Vset*)

have $\mathfrak{K}$-*def*: $\mathfrak{K} = \mathit{cf\text{-}const}\ \mathfrak{B}\ (\mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f})\ \mathfrak{a}$
by (*rule cf-const-if-HomCod-is-cat-1*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
have $\mathfrak{G}\mathfrak{K}$-*def*: $\mathfrak{G} \circ_{CF} \mathfrak{K} = \mathit{cf\text{-}const}\ \mathfrak{B}\ \mathfrak{A}\ (\mathfrak{G}(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{a}|\!))$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-1-components*(*1*) $\mathfrak{K}$-*def cat-cs-simps*
cs-intro: *V-cs-intros cat-cs-intros*
)

interpret ε: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \langle \mathfrak{G} \circ_{CF} \mathfrak{K}\rangle\ \mathfrak{T}\ \varepsilon$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

show $\varepsilon : \mathfrak{G}(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{a}|\!) <_{CF.lim} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof(*intro is-cat-limitI is-cat-coneI*)

show ε : *cf-const* $\mathfrak{B}\ \mathfrak{A}\ (\mathfrak{G}(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{a}|\!)) \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by (*rule* ε.*ntcf-rKe.is-ntcf-axioms*[*unfolded* $\mathfrak{G}\mathfrak{K}$-*def*])

fix $u'\ r'$ **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

interpret u': *is-cat-cone* $\alpha\ r'\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{T}\ u'$ **by** (*rule prems*)

have $\mathfrak{G}$-*def*: $\mathfrak{G} = \mathit{cf\text{-}const}\ (\mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f})\ \mathfrak{A}\ (\mathfrak{G}(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{a}|\!))$
by (*rule cf-const-if-HomDom-is-cat-1*[*OF* ε.*Ran.is-functor-axioms*])

from *prems* **have** *const-r'*: *cf-const* $(\mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f})\ \mathfrak{A}\ r' : \mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-lim-cs-intros cat-cs-intros*
)

have *cf-comp-cf-const-r-*$\mathfrak{K}$-*def*:
cf-const $(\mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f})\ \mathfrak{A}\ r' \circ_{CF} \mathfrak{K} = \mathit{cf\text{-}const}\ \mathfrak{B}\ \mathfrak{A}\ r'$
by
(

```
        cs-concl
          cs-simp: cat-cs-simps 𝔎-def
          cs-intro: cat-cs-intros cat-lim-cs-intros
      )

    from ε.cat-rKe-unique[
        OF const-r', unfolded cf-comp-cf-const-r-𝔎-def, OF u'.is-ntcf-axioms
        ]
    obtain σ
      where σ: σ : cf-const (cat-1 𝔞 𝔣) 𝔄 r' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄
        and u'-def: u' = ε ⋆_NTCF (σ ∘_NTCF-CF 𝔎)
        and unique-σ: ⋀σ'.
          ⟦
            σ' : cf-const (cat-1 𝔞 𝔣) 𝔄 r' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄;
            u' = ε ⋆_NTCF (σ' ∘_NTCF-CF 𝔎)
          ⟧ ⟹ σ' = σ
      by auto

    interpret σ: is-ntcf α ‹cat-1 𝔞 𝔣› 𝔄 ‹cf-const (cat-1 𝔞 𝔣) 𝔄 r'› 𝔊 σ
      by (rule σ)

    show ∃!f'. f' : r' ↦_𝔄 𝔊(‖ObjMap‖)(‖𝔞‖) ∧ u' = ε ⋆_NTCF ntcf-const 𝔅 𝔄 f'
    proof(intro ex1I conjI; (elim conjE)?)
      fix f' assume prems':
        f' : r' ↦_𝔄 𝔊(‖ObjMap‖)(‖𝔞‖) u' = ε ⋆_NTCF ntcf-const 𝔅 𝔄 f'
      from prems'(1) have ntcf-const (cat-1 𝔞 𝔣) 𝔄 f' :
        cf-const (cat-1 𝔞 𝔣) 𝔄 r' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄
        by (subst 𝔊-def)
          (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      moreover with prems'(1) have u' = ε ⋆_NTCF (ntcf-const (cat-1 𝔞 𝔣) 𝔄 f' ∘_NTCF-CF 𝔎)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps prems'(2) 𝔎-def cs-intro: cat-cs-intros
          )
      ultimately have σ-def: σ = ntcf-const (cat-1 𝔞 𝔣) 𝔄 f'
        by (auto simp: unique-σ[symmetric])
      show f' = σ(‖NTMap‖)(‖𝔞‖)
        by (cs-concl cs-simp: cat-cs-simps σ-def cs-intro: cat-cs-intros)
    qed (cs-concl cs-simp: cat-cs-simps u'-def 𝔎-def cs-intro: cat-cs-intros)+

  qed (cs-concl cs-simp: 𝔎-def cs-intro: cat-cs-intros)

qed

lemma cat-lKe-is-cat-colimit:
  assumes η : 𝔗 ↦_CF.lKeα 𝔉 ∘_CF 𝔎 : 𝔅 ↦_C cat-1 𝔞 𝔣 ↦_C 𝔄
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
  shows η : 𝔗 >_CF.colim 𝔉(‖ObjMap‖)(‖𝔞‖) : 𝔅 ↦↦_Cα 𝔄
proof–
  interpret η: is-cat-lKe α 𝔅 ‹cat-1 𝔞 𝔣› 𝔄 𝔎 𝔗 𝔉 η by (rule assms(1))
  from cat-1-components(1) have 𝔞: 𝔞 ∈_∘ Vset α
    by (auto simp: η.AG.HomCod.cat-in-Obj-in-Vset)
  from cat-1-components(2) have 𝔣: 𝔣 ∈_∘ Vset α
    by (auto simp: η.AG.HomCod.cat-in-Arr-in-Vset)
  show ?thesis
    by
      (
```

```
      rule is-cat-limit.is-cat-colimit-op
        [
          OF cat-rKe-is-cat-limit[
            OF
              η.is-cat-rKe-op[unfolded η.AG.cat-1-op[OF 𝔞 𝔣]]
              η.ntcf-lKe.NTDom.is-functor-op
            ],
          unfolded cat-op-simps
        ]
    )
qed

lemma cat-limit-is-rKe:
  — The statement of the theorem is similar to the statement of a part of Theorem 1 in Chapter X-7 in
[9] or Proposition 6.5.1 in [14].
  assumes ε : 𝔊(|ObjMap|)(|𝔞|) <_{CF.lim} 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
    and 𝔎 : 𝔅 ↦↦_{Cα} cat-1 𝔞 𝔣
    and 𝔊 : cat-1 𝔞 𝔣 ↦↦_{Cα} 𝔄
  shows ε : 𝔊 ∘_{CF} 𝔎 ↦_{CF.rKeα} 𝔗 : 𝔅 ↦_C cat-1 𝔞 𝔣 ↦_C 𝔄
proof–

  interpret ε: is-cat-limit α 𝔅 𝔄 𝔗 ‹𝔊(|ObjMap|)(|𝔞|)› ε by (rule assms)
  interpret 𝔎: is-functor α 𝔅 ‹cat-1 𝔞 𝔣› 𝔎 by (rule assms(2))
  interpret 𝔊: is-functor α ‹cat-1 𝔞 𝔣› 𝔄 𝔊 by (rule assms(3))

  show ?thesis
  proof(rule is-cat-rKeI′)

    note 𝔎-def = cf-const-if-HomCod-is-cat-1[OF assms(2)]
    note 𝔊-def = cf-const-if-HomDom-is-cat-1[OF assms(3)]

    have 𝔊𝔎-def: 𝔊 ∘_{CF} 𝔎 = cf-const 𝔅 𝔄 (𝔊(|ObjMap|)(|𝔞|))
      by (subst 𝔎-def, use nothing in ‹subst 𝔊-def›)
        (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

    show ε : 𝔊 ∘_{CF} 𝔎 ↦_{CF} 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps 𝔊𝔎-def cs-intro: cat-cs-intros
        )

    fix 𝔊′ ε′ assume prems:
      𝔊′ : cat-1 𝔞 𝔣 ↦↦_{Cα} 𝔄
      ε′ : 𝔊′ ∘_{CF} 𝔎 ↦_{CF} 𝔗 : 𝔅 ↦↦_{Cα} 𝔄

    interpret is-functor α ‹cat-1 𝔞 𝔣› 𝔄 𝔊′ by (rule prems(1))

    note 𝔊′-def = cf-const-if-HomDom-is-cat-1[OF prems(1)]

    from prems(2) have ε′:
      ε′ : cf-const 𝔅 𝔄 (𝔊′(|ObjMap|)(|𝔞|)) ↦_{CF} 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
      unfolding 𝔎-def
      by (subst (asm) 𝔊′-def)
        (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from prems(2) have ε′ : 𝔊′(|ObjMap|)(|𝔞|) <_{CF.cone} 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
      by (intro is-cat-coneI ε′) (cs-concl cs-intro: cat-cs-intros)+
```

```
    from ε.cat-lim-ua-fo[OF this] obtain f'
      where f': f' : 𝔊'(|ObjMap|)(|𝔞|) ↦_𝔄 𝔊(|ObjMap|)(|𝔞|)
        and ε-def: ε' = ε ∙_NTCF ntcf-const 𝔅 𝔄 f'
        and unique-f':
          ⟦
            f'' : 𝔊'(|ObjMap|)(|𝔞|) ↦_𝔄 𝔊(|ObjMap|)(|𝔞|);
            ε' = ε ∙_NTCF ntcf-const 𝔅 𝔄 f''
          ⟧ ⟹ f'' = f'
        for f''
      by metis

    show ∃!σ.
      σ : 𝔊' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄 ∧ ε' = ε ∙_NTCF (σ ∘_NTCF-CF 𝔎)
    proof(intro ex1I conjI; (elim conjE)?)
      from f' show
        ntcf-const (cat-1 𝔞 𝔣) 𝔄 f' : 𝔊' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄
        by (subst 𝔊'-def, use nothing in ‹subst 𝔊-def›)
          (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      with f' show ε' = ε ∙_NTCF (ntcf-const (cat-1 𝔞 𝔣) 𝔄 f' ∘_NTCF-CF 𝔎)
        by (cs-concl cs-simp: cat-cs-simps ε-def 𝔎-def cs-intro: cat-cs-intros)
      fix σ assume prems:
        σ : 𝔊' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄
        ε' = ε ∙_NTCF (σ ∘_NTCF-CF 𝔎)
      interpret σ: is-ntcf α ‹cat-1 𝔞 𝔣› 𝔄 𝔊' 𝔊 σ by (rule prems(1))
      have σ(|NTMap|)(|𝔞|) : 𝔊'(|ObjMap|)(|𝔞|) ↦_𝔄 𝔊(|ObjMap|)(|𝔞|)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      moreover have ε' = ε ∙_NTCF ntcf-const 𝔅 𝔄 (σ(|NTMap|)(|𝔞|))
        by
          (
            cs-concl
              cs-simp: cat-cs-simps prems(2) 𝔎-def cs-intro: cat-cs-intros
          )
      ultimately have σ𝔞: σ(|NTMap|)(|𝔞|) = f' by (rule unique-f')
      show σ = ntcf-const (cat-1 𝔞 𝔣) 𝔄 f'
      proof(rule ntcf-eqI)
        from f' show
          ntcf-const (cat-1 𝔞 𝔣) 𝔄 f' : 𝔊' ↦_CF 𝔊 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄
          by (subst 𝔊'-def, use nothing in ‹subst 𝔊-def›)
            (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
        have dom-lhs: 𝒟∘ (σ(|NTMap|)) = cat-1 𝔞 𝔣(|Obj|)
          by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro:cat-cs-intros)
        have dom-rhs: 𝒟∘ (ntcf-const (cat-1 𝔞 𝔣) 𝔄 f'(|NTMap|)) = cat-1 𝔞 𝔣(|Obj|)
          by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro:cat-cs-intros)
        show σ(|NTMap|) = ntcf-const (cat-1 𝔞 𝔣) 𝔄 f'(|NTMap|)
        proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
          fix a assume prems: a ∈∘ cat-1 𝔞 𝔣(|Obj|)
          then have a-def: a = 𝔞 unfolding cat-1-components by simp
          from f' show σ(|NTMap|)(|a|) = ntcf-const (cat-1 𝔞 𝔣) 𝔄 f'(|NTMap|)(|a|)
            unfolding a-def σ𝔞
            by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
        qed (auto intro: cat-cs-intros)
      qed (simp-all add: prems)
    qed

  qed (auto simp: assms)

qed
```

```
lemma cat-colimit-is-lKe:
  assumes η : 𝔗 >_CF.colim 𝔉(|ObjMap|)(|𝔞|) : 𝔅 ↦↦_Cα 𝔄
    and 𝔎 : 𝔅 ↦↦_Cα cat-1 𝔞 𝔣
    and 𝔉 : cat-1 𝔞 𝔣 ↦↦_Cα 𝔄
  shows η : 𝔗 ↦_CF.lKeα 𝔉 ∘_CF 𝔎 : 𝔅 ↦_C cat-1 𝔞 𝔣 ↦_C 𝔄
proof–
  interpret η: is-cat-colimit α 𝔅 𝔄 𝔗 ‹𝔉(|ObjMap|)(|𝔞|)› η
    by (rule assms(1))
  interpret 𝔎: is-functor α 𝔅 ‹cat-1 𝔞 𝔣› 𝔎 by (rule assms(2))
  interpret 𝔉: is-functor α ‹cat-1 𝔞 𝔣› 𝔄 𝔉 by (rule assms(3))
  from cat-1-components(1) have 𝔞: 𝔞 ∈∘ Vset α
    by (auto simp: 𝔎.HomCod.cat-in-Obj-in-Vset)
  from cat-1-components(2) have 𝔣: 𝔣 ∈∘ Vset α
    by (auto simp: 𝔎.HomCod.cat-in-Arr-in-Vset)
  have 𝔉𝔞: 𝔉(|ObjMap|)(|𝔞|) = op-cf 𝔉(|ObjMap|)(|𝔞|) unfolding cat-op-simps by simp
  note cat-1-op = η.cat-1-op[OF 𝔞 𝔣]
  show ?thesis
    by
      (
        rule is-cat-rKe.is-cat-lKe-op
          [
            OF cat-limit-is-rKe
              [
                OF
                  η.is-cat-limit-op[unfolded 𝔉𝔞]
                  𝔎.is-functor-op[unfolded cat-1-op]
                  𝔉.is-functor-op[unfolded cat-1-op]
              ],
            unfolded cat-op-simps cat-1-op
          ]
      )
qed
```

14.6.2 Adjoints

```
lemma (in is-cf-adjunction) cf-adjunction-counit-is-rKe:
```
— The statement of the theorem is similar to the statement of a part of Theorem 2 in Chapter X-7 in [9] or Proposition 6.5.2 in [14]. The proof follows (approximately) the proof in [14].
```
  shows ε_C Φ : 𝔉 ∘_CF 𝔊 ↦_CF.rKeα cf-id 𝔇 : 𝔇 ↦_C ℭ ↦_C 𝔇
proof–

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝒵-Limit-αω 𝒵-ω-αω 𝒵-def 𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  note exp-adj = cf-adj-exp-cf-cat-exp-cf-cat[OF β αβ R.category-axioms]

  let ?η = ‹η_C Φ›
  let ?ε = ‹ε_C Φ›
  let ?𝔇η = ‹exp-cat-ntcf α 𝔇 ?η›
  let ?𝔇𝔉 = ‹exp-cat-cf α 𝔇 𝔉›
  let ?𝔇𝔊 = ‹exp-cat-cf α 𝔇 𝔊›
  let ?𝔇𝔇 = ‹cat-FUNCT α 𝔇 𝔇›
  let ?ℭ𝔇 = ‹cat-FUNCT α ℭ 𝔇›
  let ?adj-𝔇η = ‹cf-adjunction-of-unit β ?𝔇𝔊 ?𝔇𝔉 ?𝔇η›

  interpret 𝔇η: is-cf-adjunction β ?ℭ𝔇 ?𝔇𝔇 ?𝔇𝔊 ?𝔇𝔉 ?adj-𝔇η by (rule exp-adj)
```

```
show ?thesis
proof(intro is-cat-rKeI)
  have id-𝔇: cf-map (cf-id 𝔇) ∈∘ cat-FUNCT α 𝔇 𝔇(|Obj|)
    by
      (
        cs-concl
          cs-simp: cat-FUNCT-components(1)
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  then have exp-id-𝔇:
    exp-cat-cf α 𝔇 𝔉(|ObjMap|)(|cf-map (cf-id 𝔇)|) = cf-map 𝔉
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cs-intro: cat-cs-intros
      )
  have 𝔉: cf-map 𝔉 ∈∘ cat-FUNCT α ℭ 𝔇(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-components(1)
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  have ε: ntcf-arrow (ε_C Φ) ∈∘ ntcf-arrows α 𝔇 𝔇
    by (cs-concl cs-intro: cat-FUNCT-cs-intros adj-cs-intros)
  have 𝔇𝔇: category β (cat-FUNCT α 𝔇 𝔇)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  have ℭ𝔇: category β (cat-FUNCT α ℭ 𝔇)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)

  from
    ε 𝔉 αβ id-𝔇
    𝔇𝔇 ℭ𝔇 LR.is-functor-axioms RL.is-functor-axioms R.cat-cf-id-is-functor
    NT.is-iso-ntcf-axioms
  have ε-id-𝔇: ε_C ?adj-𝔇η(|NTMap|)(|cf-map (cf-id 𝔇)|) = ntcf-arrow ?ε
    by
      (
        cs-concl
          cs-simp:
            cat-Set-the-inverse[symmetric]
            cat-op-simps
            cat-cs-simps
            cat-FUNCT-cs-simps
            adj-cs-simps
          cs-intro:
            𝔇η.NT.iso-ntcf-is-iso-arr''
            cat-op-intros
            adj-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-prod-cs-intros
      )
 show universal-arrow-fo ?𝔇𝔊 (cf-map (cf-id 𝔇)) (cf-map 𝔉) (ntcf-arrow ?ε)
    by
      (
        rule is-cf-adjunction.cf-adjunction-counit-component-is-ua-fo[
          OF exp-adj id-𝔇, unfolded exp-id-𝔇 ε-id-𝔇
```

```
        ]
      )
  qed (cs-concl cs-intro: cat-cs-intros adj-cs-intros)+

qed

lemma (in is-cf-adjunction) cf-adjunction-unit-is-lKe:
  shows η_C Φ : cf-id ℭ ↦_{CF.lKeα} 𝔊 ∘_{CF} 𝔉 : ℭ ↦_C 𝔇 ↦_C ℭ
  by
    (
      rule is-cat-rKe.is-cat-lKe-op
        [
          OF is-cf-adjunction.cf-adjunction-counit-is-rKe
            [
              OF is-cf-adjunction-op,
              folded op-ntcf-cf-adjunction-unit op-cf-cf-id
            ],
          unfolded
            cat-op-simps ntcf-op-ntcf-op-ntcf[OF cf-adjunction-unit-is-ntcf]
        ]
    )

lemma cf-adjunction-if-lKe-preserves:
  — The statement of the theorem is similar to the statement of a part of Theorem 2 in Chapter X-7 in [9] or Proposition 6.5.2 in [14].
  assumes η : cf-id 𝔇 ↦_{CF.lKeα} 𝔉 ∘_{CF} 𝔊 : 𝔇 ↦_C ℭ ↦_C (𝔊 : 𝔇 ↦↦_C ℭ)
  shows cf-adjunction-of-unit α 𝔊 𝔉 η : 𝔊 ⇌_{CF} 𝔉 : 𝔇 ⇌⇌_{Cα} ℭ
proof–

  interpret η: is-cat-lKe-preserves α 𝔇 ℭ 𝔇 ℭ 𝔊 ‹cf-id 𝔇› 𝔉 𝔊 η
    by (rule assms)

  from η.cat-lKe-preserves interpret 𝔊η:
    is-cat-lKe α 𝔇 ℭ ℭ 𝔊 𝔊 ‹𝔊 ∘_{CF} 𝔉› ‹𝔊 ∘_{CF-NTCF} η›
    by (cs-prems cs-shallow cs-simp: cat-cs-simps)

  from
    𝔊η.cat-lKe-unique
      [
        OF η.AG.HomCod.cat-cf-id-is-functor,
        unfolded η.AG.cf-cf-comp-cf-id-left,
        OF η.AG.cf-ntcf-id-is-ntcf
      ]
  obtain ε where ε: ε : 𝔊 ∘_{CF} 𝔉 ↦_{CF} cf-id ℭ : ℭ ↦↦_{Cα} ℭ
    and ntcf-id-𝔊-def: ntcf-id 𝔊 = ε ∘_{NTCF-CF} 𝔊 ·_{NTCF} (𝔊 ∘_{CF-NTCF} η)
    by metis
  interpret ε: is-ntcf α ℭ ℭ ‹𝔊 ∘_{CF} 𝔉› ‹cf-id ℭ› ε by (rule ε)

  show ?thesis
  proof(rule counit-unit-is-cf-adjunction)

    show [cat-cs-simps]: ε ∘_{NTCF-CF} 𝔊 ·_{NTCF} (𝔊 ∘_{CF-NTCF} η) = ntcf-id 𝔊
      by (rule ntcf-id-𝔊-def[symmetric])

    have η-def: η = (ntcf-id 𝔉 ∘_{NTCF-CF} 𝔊) ·_{NTCF} η
      by
        (
          cs-concl cs-shallow
```

```
      cs-simp: cat-cs-simps ntcf-id-cf-comp[symmetric]
      cs-intro: cat-cs-intros
  )
note [cat-cs-simps] = this[symmetric]

let ?𝔉𝜀𝔊 = ‹𝔉 ∘_CF−NTCF 𝜀 ∘_NTCF−CF 𝔊›
let ?𝜂𝔉𝔊 = ‹𝜂 ∘_NTCF−CF 𝔉 ∘_NTCF−CF 𝔊›
let ?𝔉𝔊𝜂 = ‹𝔉 ∘_CF 𝔊 ∘_CF−NTCF 𝜂›

have (?𝔉𝜀𝔊 ⊙_NTCF ?𝜂𝔉𝔊) ⊙_NTCF 𝜂 = (?𝔉𝜀𝔊 ⊙_NTCF ?𝔉𝔊𝜂) ⊙_NTCF 𝜂
proof(rule ntcf-eqI)
  have dom-lhs: 𝒟∘ (((?𝔉𝜀𝔊 ⊙_NTCF ?𝜂𝔉𝔊) ⊙_NTCF 𝜂)(|NTMap|)) = 𝔇(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  have dom-rhs: 𝒟∘ (((?𝔉𝜀𝔊 ⊙_NTCF ?𝔉𝔊𝜂) ⊙_NTCF 𝜂)(|NTMap|)) = 𝔇(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  note is-ntcf.ntcf-Comp-commute[cat-cs-simps del]
  note category.cat-Comp-assoc[cat-cs-simps del]
  show
    ((?𝔉𝜀𝔊 ⊙_NTCF ?𝜂𝔉𝔊) ⊙_NTCF 𝜂)(|NTMap|) =
      ((?𝔉𝜀𝔊 ⊙_NTCF ?𝔉𝔊𝜂) ⊙_NTCF 𝜂)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume a ∈∘ 𝔇(|Obj|)
    then show
      ((?𝔉𝜀𝔊 ⊙_NTCF ?𝜂𝔉𝔊) ⊙_NTCF 𝜂)(|NTMap|)(|a|) =
        ((?𝔉𝜀𝔊 ⊙_NTCF ?𝔉𝔊𝜂) ⊙_NTCF 𝜂)(|NTMap|)(|a|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps 𝜂.ntcf-lKe.ntcf-Comp-commute[symmetric]
            cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros)+
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
also have ... = (ntcf-id 𝔉 ∘_NTCF−CF 𝔊) ⊙_NTCF 𝜂
  by
    (
      cs-concl cs-shallow
        cs-simp:
          cat-cs-simps
          cf-comp-cf-ntcf-comp-assoc
          cf-ntcf-comp-ntcf-cf-comp-assoc
          cf-ntcf-comp-ntcf-vcomp[symmetric]
        cs-intro: cat-cs-intros
    )
also have ... = 𝜂 by (cs-concl cs-simp: cat-cs-simps)
finally have (?𝔉𝜀𝔊 ⊙_NTCF ?𝜂𝔉𝔊) ⊙_NTCF 𝜂 = 𝜂 by simp
then have 𝜂-def': 𝜂 = (𝔉 ∘_CF−NTCF 𝜀 ⊙_NTCF (𝜂 ∘_NTCF−CF 𝔉) ∘_NTCF−CF 𝔊) ⊙_NTCF 𝜂
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps ntcf-vcomp-ntcf-cf-comp[symmetric]
        cs-intro: cat-cs-intros
    )+

have 𝔉𝜀𝜂𝔉: 𝔉 ∘_CF−NTCF 𝜀 ⊙_NTCF (𝜂 ∘_NTCF−CF 𝔉) : 𝔉 ↦_CF 𝔉 : ℭ ↦↦_Cα 𝔇
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

from 𝜂.cat-lKe-unique[OF 𝜂.Lan.is-functor-axioms 𝜂.ntcf-lKe.is-ntcf-axioms]
```

```
    obtain σ where
      [[ σ' : 𝔉 ↦_CF 𝔉 : ℭ ↦↦_Cα 𝔇; η = σ' ∘_NTCF-CF 𝔊 •_NTCF η ]] ⟹ σ' = σ
      for σ'
      by metis

    from this[OF η.Lan.cf-ntcf-id-is-ntcf η-def] this[OF 𝔉εη𝔉 η-def'] show
      𝔉 ∘_CF-NTCF ε •_NTCF (η ∘_NTCF-CF 𝔉) = ntcf-id 𝔉
      by simp

  qed (cs-concl cs-intro: cat-cs-intros)+

qed

lemma cf-adjunction-if-rKe-preserves:
  assumes ε : 𝔉 ∘_CF 𝔊 ↦_CF.rKeα cf-id 𝔇 : 𝔇 ↦_C ℭ ↦_C (𝔊 : 𝔇 ↦↦_C ℭ)
  shows cf-adjunction-of-counit α 𝔉 𝔊 ε : 𝔉 ⇌_CF 𝔊 : ℭ ⇌⇌_Cα 𝔇
proof–
  interpret ε: is-cat-rKe-preserves α 𝔇 ℭ 𝔇 ℭ 𝔊 ‹cf-id 𝔇› 𝔉 𝔊 ε
    by (rule assms)
  have op-cf (cf-id 𝔇) = cf-id (op-cat 𝔇) unfolding cat-op-simps by simp
  show ?thesis
    by
      (
        rule is-cf-adjunction.is-cf-adjunction-op
          [
            OF cf-adjunction-if-lKe-preserves[
              OF ε.is-cat-rKe-preserves-op[unfolded op-cf-cf-id]
              ],
            folded cf-adjunction-of-counit-def,
            unfolded cat-op-simps
          ]
      )
qed
```

15 Pointwise Kan extensions

15.1 Pointwise Kan extensions

The following subsection is based on elements of the content of section 6.3 in [14] and Chapter X-5 in [9].

locale *is-cat-pw-rKe* = *is-cat-rKe* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ ε
  for α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ ε +
  assumes *cat-pw-rKe-preserved*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!) \Longrightarrow$
    ε :
      $\mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T}$:
      $\mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} (Hom_{O.C\alpha}\mathfrak{A}(a,-) : \mathfrak{A} \mapsto\mapsto_{C} cat\text{-}Set\ \alpha)$

syntax *-is-cat-pw-rKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  (
    $\langle$(- :/ - $\circ_{CF}$ - $\mapsto_{CF.rKe.pw1}$ - :/ - $\mapsto_{C}$ - $\mapsto_{C}$ -)$\rangle$
    [51, 51, 51, 51, 51, 51, 51] 51
)
syntax-consts *-is-cat-pw-rKe* $\rightleftharpoons$ *is-cat-pw-rKe*
translations $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe.pw\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} \mathfrak{A} \rightleftharpoons$
  CONST is-cat-pw-rKe α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ ε

locale *is-cat-pw-lKe* = *is-cat-lKe* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ η
  for α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ η +
  assumes *cat-pw-lKe-preserved*: $a \in_{\circ} op\text{-}cat\ \mathfrak{A}(\!|Obj|\!) \Longrightarrow$
    op-ntcf η :
      op-cf $\mathfrak{F} \circ_{CF}$ *op-cf* $\mathfrak{K} \mapsto_{CF.rKe\alpha}$ *op-cf* $\mathfrak{T}$:
      op-cat $\mathfrak{B} \mapsto_{C}$ *op-cat* $\mathfrak{C} \mapsto_{C} (Hom_{O.C\alpha}\mathfrak{A}(-,a) : op\text{-}cat\ \mathfrak{A} \mapsto\mapsto_{C} cat\text{-}Set\ \alpha)$

syntax *-is-cat-pw-lKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  (
    $\langle$(- :/ - $\mapsto_{CF.lKe.pw1}$ - $\circ_{CF}$ - :/ - $\mapsto_{C}$ - $\mapsto_{C}$ -)$\rangle$
    [51, 51, 51, 51, 51, 51, 51] 51
)
syntax-consts *-is-cat-pw-lKe* $\rightleftharpoons$ *is-cat-pw-lKe*
translations $\eta : \mathfrak{T} \mapsto_{CF.lKe.pw\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} \mathfrak{A} \rightleftharpoons$
  CONST is-cat-pw-lKe α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ η

lemma (**in** *is-cat-pw-rKe*) *cat-pw-rKe-preserved'*[*cat-Kan-cs-intros*]:
  assumes $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
    and $\mathfrak{A}' = \mathfrak{A}$
    and $\mathfrak{H}' = Hom_{O.C\alpha}\mathfrak{A}(a,-)$
    and $\mathfrak{E}' = cat\text{-}Set\ \alpha$
  shows $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_{C} \mathfrak{C} \mapsto_{C} (\mathfrak{H}' : \mathfrak{A}' \mapsto\mapsto_{C} \mathfrak{E}')$
  using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cat-pw-rKe-preserved*)

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-rKe.cat-pw-rKe-preserved'*

lemma (**in** *is-cat-pw-lKe*) *cat-pw-lKe-preserved'*[*cat-Kan-cs-intros*]:
  assumes $a \in_{\circ} op\text{-}cat\ \mathfrak{A}(\!|Obj|\!)$
    and $\mathfrak{F}' =$ *op-cf* $\mathfrak{F}$
    and $\mathfrak{K}' =$ *op-cf* $\mathfrak{K}$
    and $\mathfrak{T}' =$ *op-cf* $\mathfrak{T}$
    and $\mathfrak{B}' =$ *op-cat* $\mathfrak{B}$
    and $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$
    and $\mathfrak{A}' =$ *op-cat* $\mathfrak{A}$
    and $\mathfrak{H}' = Hom_{O.C\alpha}\mathfrak{A}(-,a)$
    and $\mathfrak{E}' = cat\text{-}Set\ \alpha$

shows *op-ntcf* η :
$\mathfrak{F}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe\alpha} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto\mapsto_C \mathfrak{E}')$
using *assms*(*1*) **unfolding** *assms* **by** (*rule cat-pw-lKe-preserved*)

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-lKe.cat-pw-lKe-preserved'*

Rules.

lemma (**in** *is-cat-pw-rKe*) *is-cat-pw-rKe-axioms'*[*cat-Kan-cs-intros*]:
assumes $\alpha' = \alpha$
and $\mathfrak{G}' = \mathfrak{G}$
and $\mathfrak{K}' = \mathfrak{K}$
and $\mathfrak{T}' = \mathfrak{T}$
and $\mathfrak{B}' = \mathfrak{B}$
and $\mathfrak{A}' = \mathfrak{A}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\varepsilon : \mathfrak{G}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe.pw_{\alpha}'} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-pw-rKe-axioms*)

mk-ide rf *is-cat-pw-rKe-def*[*unfolded is-cat-pw-rKe-axioms-def*]
|*intro is-cat-pw-rKeI*|
|*dest is-cat-pw-rKeD*[*dest*]|
|*elim is-cat-pw-rKeE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-rKeD*(*1*)

lemma (**in** *is-cat-pw-lKe*) *is-cat-pw-lKe-axioms'*[*cat-Kan-cs-intros*]:
assumes $\alpha' = \alpha$
and $\mathfrak{F}' = \mathfrak{F}$
and $\mathfrak{K}' = \mathfrak{K}$
and $\mathfrak{T}' = \mathfrak{T}$
and $\mathfrak{B}' = \mathfrak{B}$
and $\mathfrak{A}' = \mathfrak{A}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\eta : \mathfrak{T}' \mapsto_{CF.lKe.pw_{\alpha}'} \mathfrak{F}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-pw-lKe-axioms*)

mk-ide rf *is-cat-pw-lKe-def*[*unfolded is-cat-pw-lKe-axioms-def*]
|*intro is-cat-pw-lKeI*|
|*dest is-cat-pw-lKeD*[*dest*]|
|*elim is-cat-pw-lKeE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-lKeD*(*1*)

Duality.

lemma (**in** *is-cat-pw-rKe*) *is-cat-pw-lKe-op*:
op-ntcf ε :
op-cf $\mathfrak{T} \mapsto_{CF.lKe.pw\alpha}$ *op-cf* $\mathfrak{G} \circ_{CF}$ *op-cf* $\mathfrak{K}$:
op-cat $\mathfrak{B} \mapsto_C$ *op-cat* $\mathfrak{C} \mapsto_C$ *op-cat* $\mathfrak{A}$
proof(*intro is-cat-pw-lKeI*, *unfold cat-op-simps*)
fix *a* **assume** *prems*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
from *cat-pw-rKe-preserved*[*OF prems*] *prems* **show**
ε :
$\mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T}$:
$\mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (Hom_{O.C\alpha}$ *op-cat* $\mathfrak{A}(-,a) : \mathfrak{A} \mapsto\mapsto_C$ *cat-Set* $\alpha)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*)
qed (*cs-concl* **cs-shallow cs-intro**: *cat-op-intros*)

lemma (**in** *is-cat-pw-rKe*) *is-cat-pw-lKe-op'*[*cat-op-intros*]:

```
  assumes 𝔗′ = op-cf 𝔗
    and 𝔊′ = op-cf 𝔊
    and 𝔎′ = op-cf 𝔎
    and 𝔅′ = op-cat 𝔅
    and 𝔄′ = op-cat 𝔄
    and ℭ′ = op-cat ℭ
  shows op-ntcf ε : 𝔗′ ↦CF.lKe.pwα 𝔊′ ∘CF 𝔎′ : 𝔅′ ↦C ℭ′ ↦C 𝔄′
  unfolding assms by (rule is-cat-pw-lKe-op)

lemmas [cat-op-intros] = is-cat-pw-rKe.is-cat-pw-lKe-op′

lemma (in is-cat-pw-lKe) is-cat-pw-rKe-op:
  op-ntcf η :
    op-cf 𝔉 ∘CF op-cf 𝔎 ↦CF.rKe.pwα op-cf 𝔗 :
    op-cat 𝔅 ↦C op-cat ℭ ↦C op-cat 𝔄
proof(intro is-cat-pw-rKeI, unfold cat-op-simps)
  fix a assume prems: a ∈∘ 𝔄⦇Obj⦈
  from cat-pw-lKe-preserved[unfolded cat-op-simps, OF prems] prems show
    op-ntcf η :
      op-cf 𝔉 ∘CF op-cf 𝔎 ↦CF.rKeα op-cf 𝔗 :
      op-cat 𝔅 ↦C op-cat ℭ ↦C
      (Hom_O.Cα op-cat 𝔄(a,–) : op-cat 𝔄 ↦↦C cat-Set α)
    by (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros)
qed (cs-concl cs-shallow cs-intro: cat-op-intros)

lemma (in is-cat-pw-lKe) is-cat-pw-lKe-op′[cat-op-intros]:
  assumes 𝔗′ = op-cf 𝔗
    and 𝔉′ = op-cf 𝔉
    and 𝔎′ = op-cf 𝔎
    and 𝔅′ = op-cat 𝔅
    and 𝔄′ = op-cat 𝔄
    and ℭ′ = op-cat ℭ
  shows op-ntcf η : 𝔉′ ∘CF 𝔎′ ↦CF.rKe.pwα 𝔗′ : 𝔅′ ↦C ℭ′ ↦C 𝔄′
  unfolding assms by (rule is-cat-pw-rKe-op)

lemmas [cat-op-intros] = is-cat-pw-lKe.is-cat-pw-lKe-op′
```

15.2 Lemma X.5: *L-10-5-N*

This subsection and several further subsections (15.2-15.8) expose definitions that are used in the proof of the technical lemma that was used in the proof of Theorem 3 from Chapter X-5 in [9].

```
definition L-10-5-N :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where L-10-5-N α β 𝔗 𝔎 c =
    [
      (
        λa∈∘𝔗⦇HomCod⦈⦇Obj⦈.
          cf-nt α β 𝔎⦇ObjMap⦈⦇cf-map (Hom_O.Cα𝔗⦇HomCod⦈(a,–) ∘CF 𝔗), c⦈•
      ),
      (
        λf∈∘𝔗⦇HomCod⦈⦇Arr⦈.
          cf-nt α β 𝔎⦇ArrMap⦈⦇
            ntcf-arrow (Hom_A.Cα𝔗⦇HomCod⦈(f,–) ∘NTCF−CF 𝔗), 𝔎⦇HomCod⦈⦇CId⦈⦇c⦈
            ⦈•
      ),
      op-cat (𝔗⦇HomCod⦈),
      cat-Set β
```

]$_\circ$

Components.

lemma *L-10-5-N-components*:
shows *L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*ObjMap*|) =
(
$\lambda a \in_\circ \mathfrak{T}(|HomCod|)(|Obj|).$
cf-nt α β $\mathfrak{K}(|ObjMap|)(|$*cf-map* ($Hom_{O.C\alpha}\mathfrak{T}(|HomCod|)(a,-) \circ_{CF} \mathfrak{T}$), *c*|)$_\bullet$
)
and *L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*ArrMap*|) =
(
$\lambda f \in_\circ \mathfrak{T}(|HomCod|)(|Arr|).$
cf-nt α β $\mathfrak{K}(|ArrMap|)(|$
ntcf-arrow ($Hom_{A.C\alpha}\mathfrak{T}(|HomCod|)(f,-) \circ_{NTCF-CF} \mathfrak{T}$), $\mathfrak{K}(|HomCod|)(|CId|)(|c|)$
|)$_\bullet$
)
and *L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*HomDom*|) = *op-cat* ($\mathfrak{T}(|HomCod|)$)
and *L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*HomCod*|) = *cat-Set* β
unfolding *L-10-5-N-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)
interpretation $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *L-10-5-N-components′* = *L-10-5-N-components*[
where $\mathfrak{T}$=$\mathfrak{T}$ **and** $\mathfrak{K}$=$\mathfrak{K}$, *unfolded cat-cs-simps*
]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-N-components′*(*3*,*4*)

end

15.2.1 Object map

mk-VLambda *L-10-5-N-components*(*1*)
|*vsv L-10-5-N-ObjMap-vsv*[*cat-Kan-cs-intros*]|

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ *c*
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

mk-VLambda *L-10-5-N-components′*(*1*)[*OF* $\mathfrak{K}$ $\mathfrak{T}$]
|*vdomain L-10-5-N-ObjMap-vdomain*[*cat-Kan-cs-simps*]|
|*app L-10-5-N-ObjMap-app*[*cat-Kan-cs-simps*]|

end

15.2.2 Arrow map

mk-VLambda *L-10-5-N-components*(*2*)
|*vsv L-10-5-N-ArrMap-vsv*[*cat-Kan-cs-intros*]|

context
  fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$ c
  assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

mk-VLambda *L-10-5-N-components'*(*2*)[*OF* $\mathfrak{K}$ $\mathfrak{T}$]
  |*vdomain L-10-5-N-ArrMap-vdomain*[*cat-Kan-cs-simps*]|
  |*app L-10-5-N-ArrMap-app*[*cat-Kan-cs-simps*]|

end

15.2.3 *L-10-5-N* is a functor

lemma *L-10-5-N-is-functor*:
  assumes $\mathcal{Z}$ β
    and $\alpha \in_\circ \beta$
    and $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
  shows *L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c : *op-cat* $\mathfrak{A} \mapsto\mapsto_{C\beta}$ *cat-Set* β
proof–

  let *?FUNCT* = ‹$\lambda\mathfrak{A}$. *cat-FUNCT* α $\mathfrak{A}$ (*cat-Set* α)›

  interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))

  interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*3*))
  interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*4*))

  from *assms*(*2*) **interpret** *FUNCT-*$\mathfrak{B}$: *tiny-category* β ‹*?FUNCT* $\mathfrak{B}$›
    by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)

  interpret $\beta\mathfrak{K}$: *is-tiny-functor* β $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$
    by (*rule is-functor.cf-is-tiny-functor-if-ge-Limit*)
      (*use assms*(*2*) **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+
  interpret $\beta\mathfrak{T}$: *is-tiny-functor* β $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$
    by (*rule is-functor.cf-is-tiny-functor-if-ge-Limit*)
      (*use assms*(*2*) **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+

  from *assms*(*2*) **interpret** *cf-nt*:
    is-functor β ‹*?FUNCT* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$› ‹*cat-Set* β› ‹*cf-nt* α β $\mathfrak{K}$›
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

  show *?thesis*
  proof(*intro is-functorI'*)

    show *vfsequence* (*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c) **unfolding** *L-10-5-N-def* **by** *simp*
    show *vcard* (*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c) = $\mathcal{4}_{\mathbb{N}}$
      unfolding *L-10-5-N-def* **by** (*simp add*: *nat-omega-simps*)
    show *vsv* (*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c(|*ObjMap*|))
      by (*cs-concl* **cs-shallow cs-intro**: *cat-Kan-cs-intros*)
    from *assms*(*3,4*) **show** *vsv* (*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c(|*ArrMap*|))
      by (*cs-concl* **cs-shallow cs-intro**: *cat-Kan-cs-intros*)
    from *assms* **show** $\mathcal{D}_\circ$ (*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c(|*ObjMap*|)) = *op-cat* $\mathfrak{A}$(|*Obj*|)
      by
        (

```
      cs-concl cs-shallow
        cs-simp: cat-Kan-cs-simps cat-op-simps cs-intro: cat-cs-intros
    )
show ℛ⊙ (L-10-5-N α β 𝔗 𝔎 c(|ObjMap|)) ⊆⊙ cat-Set β(|Obj|)
  unfolding L-10-5-N-components'[OF 𝔎.is-functor-axioms 𝔗.is-functor-axioms]
proof(rule vrange-VLambda-vsubset)
  fix a assume prems: a ∈⊙ 𝔄(|Obj|)
  from prems assms show
    cf-nt α β 𝔎(|ObjMap|)(|cf-map (Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗), c|)• ∈⊙
      cat-Set β(|Obj|)
    by
      (
        cs-concl
          cs-simp: cat-Set-components(1) cat-cs-simps  cat-FUNCT-cs-simps
          cs-intro:
            cat-cs-intros FUNCT-𝔙.cat-Hom-in-Vset cat-FUNCT-cs-intros
      )
qed

from assms show 𝒟⊙ (L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)) = op-cat 𝔄(|Arr|)
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-Kan-cs-simps cat-op-simps cs-intro: cat-cs-intros
    )

show L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)(|f|) :
  L-10-5-N α β 𝔗 𝔎 c(|ObjMap|)(|a|) ↦_{cat-Set β} L-10-5-N α β 𝔗 𝔎 c(|ObjMap|)(|b|)
  if f : a ↦_{op-cat 𝔄} b for a b f
  using that assms
  unfolding cat-op-simps
  by
    (
      cs-concl
        cs-simp: L-10-5-N-ArrMap-app L-10-5-N-ObjMap-app
        cs-intro: cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros
    )

show
  L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)(|g ∘_{A op-cat 𝔄} f|) =
    L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)(|g|) ∘_{A cat-Set β} L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)(|f|)
  if g : b′ ↦_{op-cat 𝔄} c′ and f : a′ ↦_{op-cat 𝔄} b′ for b′ c′ g a′ f
proof–
  from that assms(5) show ?thesis
    unfolding cat-op-simps
    by
      (
        cs-concl
          cs-intro:
            cat-cs-intros
            cat-prod-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
          cs-simp:
            cat-cs-simps
            cat-Kan-cs-simps
            cat-FUNCT-cs-simps
            cat-prod-cs-simps
```

```
        cat-op-simps
        cf-nt.cf-ArrMap-Comp[symmetric]
    )
  qed

  show
    L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)(|op-cat 𝔄(|CId|)(|a|)|) =
      cat-Set β(|CId|)(|L-10-5-N α β 𝔗 𝔎 c(|ObjMap|)(|a|)|)
    if a ∈∘ op-cat 𝔄(|Obj|) for a
  proof–
    note [cat-cs-simps] =
      ntcf-id-cf-comp[symmetric]
      ntcf-arrow-id-ntcf-id[symmetric]
      cat-FUNCT-CId-app[symmetric]
    from that[unfolded cat-op-simps] assms show ?thesis
      by
        (
          cs-concl
            cs-intro:
              cat-cs-intros
              cat-FUNCT-cs-intros
              cat-prod-cs-intros
              cat-op-intros
            cs-simp:
              cat-FUNCT-cs-simps cat-cs-simps cat-Kan-cs-simps cat-op-simps
        )
  qed

qed (cs-concl cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)+

qed

lemma L-10-5-N-is-functor'[cat-Kan-cs-intros]:
  assumes 𝒵 β
    and α ∈∘ β
    and 𝔎 : 𝔅 ↦↦_{C α} ℭ
    and 𝔗 : 𝔅 ↦↦_{C α} 𝔄
    and c ∈∘ ℭ(|Obj|)
    and 𝔄' = op-cat 𝔄
    and 𝔅' = cat-Set β
    and β' = β
  shows L-10-5-N α β 𝔗 𝔎 c : 𝔄' ↦↦_{C β'} 𝔅'
  using assms(1–5) unfolding assms(6–8) by (rule L-10-5-N-is-functor)
```

15.3 Lemma X.5: *L-10-5-υ-arrow*

15.3.1 Definition and elementary properties

```
definition L-10-5-υ-arrow :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where L-10-5-υ-arrow 𝔗 𝔎 c τ a b =
    [
      (λf∈∘Hom (𝔎(|HomCod|)) c (𝔎(|ObjMap|)(|b|)). τ(|NTMap|)(|0, b, f|)•),
      Hom (𝔎(|HomCod|)) c (𝔎(|ObjMap|)(|b|)),
      Hom (𝔗(|HomCod|)) a (𝔗(|ObjMap|)(|b|))
    ]∘
```

Components.

```
lemma L-10-5-υ-arrow-components:
```

shows *L-10-5-v-arrow* $\mathfrak{T}$ $\mathfrak{K}$ c τ a $b(\!|ArrVal|\!)$ =
$(\lambda f \in_\circ Hom\ (\mathfrak{K}(\!|HomCod|\!))\ c\ (\mathfrak{K}(\!|ObjMap|\!)(\!|b|\!)).\ \tau(\!|NTMap|\!)(\!|0,\ b,\ f|\!)_\bullet)$
and *L-10-5-v-arrow* $\mathfrak{T}$ $\mathfrak{K}$ c τ a $b(\!|ArrDom|\!)$ = $Hom\ (\mathfrak{K}(\!|HomCod|\!))\ c\ (\mathfrak{K}(\!|ObjMap|\!)(\!|b|\!))$
and *L-10-5-v-arrow* $\mathfrak{T}$ $\mathfrak{K}$ c τ a $b(\!|ArrCod|\!)$ = $Hom\ (\mathfrak{T}(\!|HomCod|\!))\ a\ (\mathfrak{T}(\!|ObjMap|\!)(\!|b|\!))$
unfolding *L-10-5-v-arrow-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)
interpretation $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *L-10-5-v-arrow-components′* = *L-10-5-v-arrow-components*[
where $\mathfrak{T}$=$\mathfrak{T}$ **and** $\mathfrak{K}$=$\mathfrak{K}$, *unfolded cat-cs-simps*
]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-v-arrow-components′*(*2*,*3*)

end

15.3.2 Arrow value

mk-VLambda *L-10-5-v-arrow-components*(*1*)
|*vsv L-10-5-v-arrow-ArrVal-vsv*[*cat-Kan-cs-intros*]|

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

mk-VLambda *L-10-5-v-arrow-components′*(*1*)[*OF* $\mathfrak{K}$ $\mathfrak{T}$]
|*vdomain L-10-5-v-arrow-ArrVal-vdomain*[*cat-Kan-cs-simps*]|
|*app L-10-5-v-arrow-ArrVal-app*[*unfolded in-Hom-iff*]|

end

lemma *L-10-5-v-arrow-ArrVal-app′*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\!|ObjMap|\!)(\!|b|\!)$
shows *L-10-5-v-arrow* $\mathfrak{T}$ $\mathfrak{K}$ c τ a $b(\!|ArrVal|\!)(\!|f|\!) = \tau(\!|NTMap|\!)(\!|0,\ b,\ f|\!)_\bullet$
proof–
interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*2*))
from *assms*(*3*) **have** c: $c \in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
show *?thesis* **by** (*rule L-10-5-v-arrow-ArrVal-app*[*OF assms*(*1*,*2*,*3*)])
qed

15.3.3 *L-10-5-v-arrow* is an arrow

lemma *L-10-5-v-arrow-ArrVal-is-arr*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

```
    and τ' = ntcf-arrow τ
    and τ : a <CF.cone 𝔗 ∘CF c ↓CF 𝔎 : c ↓CF 𝔎 ↦↦Cα 𝔄
    and f : c ↦ℭ 𝔎(|ObjMap|)(|b|)
    and b ∈∘ 𝔅(|Obj|)
  shows L-10-5-υ-arrow 𝔗 𝔎 c τ' a b(|ArrVal|)(|f|) : a ↦𝔄 𝔗(|ObjMap|)(|b|)
proof–
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  interpret τ: is-cat-cone α a ‹c ↓CF 𝔎› 𝔄 ‹𝔗 ∘CF c ↓CF 𝔎› τ by (rule assms(4))
  from assms(5,6) show ?thesis
    unfolding assms(3)
    by
      (
        cs-concl
          cs-simp:
            cat-cs-simps
            L-10-5-υ-arrow-ArrVal-app
            cat-comma-cs-simps
            cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed

lemma L-10-5-υ-arrow-ArrVal-is-arr'[cat-Kan-cs-intros]:
  assumes 𝔎 : 𝔅 ↦↦Cα ℭ
    and 𝔗 : 𝔅 ↦↦Cα 𝔄
    and τ' = ntcf-arrow τ
    and a' = a
    and b' = 𝔗(|ObjMap|)(|b|)
    and 𝔄' = 𝔄
    and τ : a <CF.cone 𝔗 ∘CF c ↓CF 𝔎 : c ↓CF 𝔎 ↦↦Cα 𝔄
    and f : c ↦ℭ 𝔎(|ObjMap|)(|b|)
    and b ∈∘ 𝔅(|Obj|)
  shows L-10-5-υ-arrow 𝔗 𝔎 c τ' a b(|ArrVal|)(|f|) : a' ↦𝔄 b'
  using assms(1–3, 7–9)
  unfolding assms(3–6)
  by (rule L-10-5-υ-arrow-ArrVal-is-arr)
```

15.3.4 Further properties

```
lemma L-10-5-υ-arrow-is-arr:
  assumes 𝔎 : 𝔅 ↦↦Cα ℭ
    and 𝔗 : 𝔅 ↦↦Cα 𝔄
    and c ∈∘ ℭ(|Obj|)
    and τ' = ntcf-arrow τ
    and τ : a <CF.cone 𝔗 ∘CF c ↓CF 𝔎 : c ↓CF 𝔎 ↦↦Cα 𝔄
    and b ∈∘ 𝔅(|Obj|)
  shows L-10-5-υ-arrow 𝔗 𝔎 c τ' a b :
    Hom ℭ c (𝔎(|ObjMap|)(|b|)) ↦cat-Set α Hom 𝔄 a (𝔗(|ObjMap|)(|b|))
proof–
  note L-10-5-υ-arrow-components = L-10-5-υ-arrow-components'[OF assms(1,2)]
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  interpret τ: is-cat-cone α a ‹c ↓CF 𝔎› 𝔄 ‹𝔗 ∘CF c ↓CF 𝔎› τ by (rule assms(5))
  show ?thesis
  proof(intro cat-Set-is-arrI)
    show arr-Set α (L-10-5-υ-arrow 𝔗 𝔎 c τ' a b)
    proof(intro arr-SetI)
```

```
      show vfsequence (L-10-5-υ-arrow 𝔗 ℜ c τ′ a b)
        unfolding L-10-5-υ-arrow-def by simp
      show vcard (L-10-5-υ-arrow 𝔗 ℜ c τ′ a b) = 3ℕ
        unfolding L-10-5-υ-arrow-def by (simp add: nat-omega-simps)
      show
        ℛ∘ (L-10-5-υ-arrow 𝔗 ℜ c τ′ a b(|ArrVal|)) ⊆∘
          L-10-5-υ-arrow 𝔗 ℜ c τ′ a b(|ArrCod|)
        unfolding L-10-5-υ-arrow-components
      proof(intro vrange-VLambda-vsubset, unfold in-Hom-iff)
        fix f assume f : c ↦ℭ ℜ(|ObjMap|)(|b|)
        from L-10-5-υ-arrow-ArrVal-is-arr[OF assms(1,2,4,5) this assms(6)] this
        show τ′(|NTMap|)(|0, b, f|)∙ : a ↦𝔄 𝔗(|ObjMap|)(|b|)
          by
            (
              cs-prems cs-shallow
                cs-simp: L-10-5-υ-arrow-ArrVal-app′ cat-cs-simps
                cs-intro: cat-cs-intros
            )
      qed
      from assms(3,6) show L-10-5-υ-arrow 𝔗 ℜ c τ′ a b(|ArrDom|) ∈∘ Vset α
        by (cs-concl cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)
      from assms(1–3,6) τ.cat-cone-obj show
        L-10-5-υ-arrow 𝔗 ℜ c τ′ a b(|ArrCod|) ∈∘ Vset α
        by (cs-concl cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)
    qed (auto simp: L-10-5-υ-arrow-components)
  qed (simp-all add: L-10-5-υ-arrow-components)
qed

lemma L-10-5-υ-arrow-is-arr′[cat-Kan-cs-intros]:
  assumes ℜ : 𝔅 ↦↦Cα ℭ
    and 𝔗 : 𝔅 ↦↦Cα 𝔄
    and c ∈∘ ℭ(|Obj|)
    and τ′ = ntcf-arrow τ
    and τ : a <CF.cone 𝔗 ∘CF c ∘CF ℜ : c ↓CF ℜ ↦↦Cα 𝔄
    and b ∈∘ 𝔅(|Obj|)
    and A = Hom ℭ c (ℜ(|ObjMap|)(|b|))
    and B = Hom 𝔄 a (𝔗(|ObjMap|)(|b|))
    and ℭ′ = cat-Set α
  shows L-10-5-υ-arrow 𝔗 ℜ c τ′ a b : A ↦ℭ′ B
  using assms(1–6) unfolding assms(7–9) by (rule L-10-5-υ-arrow-is-arr)

lemma L-10-5-υ-cf-hom[cat-Kan-cs-simps]:
  assumes ℜ : 𝔅 ↦↦Cα ℭ
    and 𝔗 : 𝔅 ↦↦Cα 𝔄
    and c ∈∘ ℭ(|Obj|)
    and τ′ = ntcf-arrow τ
    and τ : a <CF.cone 𝔗 ∘CF c ∘CF ℜ : c ↓CF ℜ ↦↦Cα 𝔄
    and a ∈∘ 𝔄(|Obj|)
    and f : a′ ↦𝔅 b′
  shows
    L-10-5-υ-arrow 𝔗 ℜ c τ′ a b′ ∘A cat-Set α
    cf-hom ℭ [ℭ(|CId|)(|c|), ℜ(|ArrMap|)(|f|)]∘ =
      cf-hom 𝔄 [𝔄(|CId|)(|a|), 𝔗(|ArrMap|)(|f|)]∘ ∘A cat-Set α
      L-10-5-υ-arrow 𝔗 ℜ c τ′ a a′
    (is ?lhs = ?rhs)
proof–

  interpret ℜ: is-functor α 𝔅 ℭ ℜ by (rule assms(1))
```

interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(2))
interpret τ: *is-cat-cone* α a ‹$c \downarrow_{CF} \mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T} \circ_{CF} c\ {}_{O}\!\sqcap_{CF} \mathfrak{K}$› τ **by** (*rule assms*(5))

have [*cat-Kan-cs-simps*]:
  $\tau(\!|NTMap|\!)(\!|a'', b'', \mathfrak{K}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{C}} f'|\!)_\bullet$ =
    $\mathfrak{T}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{A}} \tau(\!|NTMap|\!)(\!|a', b', f'|\!)_\bullet$
  if *F-def*: $F = [[a', b', f']_\circ, [a'', b'', f'']_\circ, [g', h']_\circ]_\circ$
    and *A-def*: $A = [a', b', f']_\circ$
    and *B-def*: $B = [a'', b'', f'']_\circ$
    and *F*: $F : A \mapsto_{c \downarrow_{CF} \mathfrak{K}} B$
  for F A B a' b' f' a'' b'' f'' g' h'
proof–
  from F[*unfolded F-def A-def B-def*] *assms*(3) **have** *a'-def*: $a' = 0$
    and *a''-def*: $a'' = 0$
    and *g'-def*: $g' = 0$
    and *h'*: $h' : b' \mapsto_{\mathfrak{B}} b''$
    and *f'*: $f' : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\!|ObjMap|\!)(\!|b'|\!)$
    and *f''*: $f'' : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\!|ObjMap|\!)(\!|b''|\!)$
    and *f''-def*: $\mathfrak{K}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{C}} f' = f''$
    by *auto*
  note τ.*cat-cone-Comp-commute*[*cat-cs-simps del*]
  from
    τ.*ntcf-Comp-commute*[*OF F*]
    that(2) F g' h' f' f''
    $\mathfrak{K}$.*is-functor-axioms*
    $\mathfrak{T}$.*is-functor-axioms*
  show
    $\tau(\!|NTMap|\!)(\!|a'', b'', \mathfrak{K}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{C}} f'|\!)_\bullet$ =
      $\mathfrak{T}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{A}} \tau(\!|NTMap|\!)(\!|a', b', f'|\!)_\bullet$
    unfolding *F-def A-def B-def a'-def a''-def g'-def*
    by
      (
        cs-prems
          cs-simp: *cat-cs-simps cat-comma-cs-simps f''-def*[*symmetric*]
          cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
qed

from *assms*(3) *assms*(6,7) $\mathfrak{K}$.*HomCod.category-axioms* **have** *lhs-is-arr*:
  ?lhs : *Hom* $\mathfrak{C}$ c $(\mathfrak{K}(\!|ObjMap|\!)(\!|a'|\!)) \mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{A}$ a $(\mathfrak{T}(\!|ObjMap|\!)(\!|b'|\!))$
  unfolding *assms*(4)
  by
    (
      cs-concl **cs-intro**:
        cat-lim-cs-intros
        cat-cs-intros
        cat-Kan-cs-intros
        cat-prod-cs-intros
        cat-op-intros
)
then have *dom-lhs*: $\mathcal{D}_\circ$ ((*?lhs*)(|*ArrVal*|)) = *Hom* $\mathfrak{C}$ c $(\mathfrak{K}(\!|ObjMap|\!)(\!|a'|\!))$
  by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *assms*(3) *assms*(6,7) $\mathfrak{K}$.*HomCod.category-axioms* $\mathfrak{T}$.*HomCod.category-axioms*
have *rhs-is-arr*:
  ?rhs : *Hom* $\mathfrak{C}$ c $(\mathfrak{K}(\!|ObjMap|\!)(\!|a'|\!)) \mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{A}$ a $(\mathfrak{T}(\!|ObjMap|\!)(\!|b'|\!))$
  unfolding *assms*(4)
  by
    (

```
      cs-concl cs-intro:
        cat-lim-cs-intros
        cat-cs-intros
        cat-Kan-cs-intros
        cat-prod-cs-intros
        cat-op-intros
    )
  then have dom-rhs: 𝒟∘ ((?rhs)(|ArrVal|)) = Hom ℭ c (ℜ(|ObjMap|)(|a′|))
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
  proof(rule arr-Set-eqI)
    from lhs-is-arr show arr-Set-lhs: arr-Set α ?lhs
      by (auto dest: cat-Set-is-arrD(1))
    from rhs-is-arr show arr-Set-rhs: arr-Set α ?rhs
      by (auto dest: cat-Set-is-arrD(1))
    show ?lhs(|ArrVal|) = ?rhs(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix g assume prems: g : c ↦ℭ ℜ(|ObjMap|)(|a′|)
      from prems assms(7) have ℜf:
        ℜ(|ArrMap|)(|f|) ∘A ℭ g : c ↦ℭ ℜ(|ObjMap|)(|b′|)
        by (cs-concl cs-shallow cs-intro: cat-cs-intros)
      with assms(6,7) prems ℜ.HomCod.category-axioms 𝔗.HomCod.category-axioms
      show ?lhs(|ArrVal|)(|g|) = ?rhs(|ArrVal|)(|g|)
        by
          (
            cs-concl
              cs-intro:
                cat-lim-cs-intros
                cat-cs-intros
                cat-Kan-cs-intros
                cat-comma-cs-intros
                cat-prod-cs-intros
                cat-op-intros
                cat-1-is-arrI
              cs-simp:
                L-10-5-υ-arrow-ArrVal-app′
                cat-cs-simps
                cat-Kan-cs-simps
                cat-op-simps
                cat-FUNCT-cs-simps
                cat-comma-cs-simps
                assms(4)
          )+
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed
    (
      use lhs-is-arr rhs-is-arr in
        ‹cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros›
    )+
qed
```

15.4 Lemma X.5: *L-10-5-τ*

15.4.1 Definition and elementary properties

```
definition L-10-5-τ where L-10-5-τ 𝔗 ℜ c υ a =
  [
    (λbf∈∘c ↓CF ℜ(|Obj|). υ(|NTMap|)(|bf(|1ℕ|)|)(|ArrVal|)(|bf(|2ℕ|)|)),
```

cf-const $(c \downarrow_{CF} \mathfrak{K})$ $(\mathfrak{T}(\lfloor HomCod \rfloor))$ a,
$\mathfrak{T} \circ_{CF} c\ _O\!\bigsqcap_{CF} \mathfrak{K}$,
$c \downarrow_{CF} \mathfrak{K}$,
$(\mathfrak{T}(\lfloor HomCod \rfloor))$
$]_\circ$

Components.

lemma *L-10-5-τ-components*:
shows *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v $a(\lfloor NTMap \rfloor) =$
$(\lambda bf \in_\circ c \downarrow_{CF} \mathfrak{K}(\lfloor Obj \rfloor).\ v(\lfloor NTMap \rfloor)(\lfloor bf(\lfloor 1_{\mathbb{N}} \rfloor) \rfloor)(\lfloor ArrVal \rfloor)(\lfloor bf(\lfloor 2_{\mathbb{N}} \rfloor) \rfloor))$
and *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v $a(\lfloor NTDom \rfloor) =$ *cf-const* $(c \downarrow_{CF} \mathfrak{K})$ $(\mathfrak{T}(\lfloor HomCod \rfloor))$ a
and *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v $a(\lfloor NTCod \rfloor) = \mathfrak{T} \circ_{CF} c\ _O\!\bigsqcap_{CF} \mathfrak{K}$
and *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v $a(\lfloor NTDGDom \rfloor) = c \downarrow_{CF} \mathfrak{K}$
and *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v $a(\lfloor NTDGCod \rfloor) = (\mathfrak{T}(\lfloor HomCod \rfloor))$
unfolding *L-10-5-τ-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)
interpretation $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *L-10-5-τ-components′* = *L-10-5-τ-components*[
where $\mathfrak{T}$=$\mathfrak{T}$ **and** $\mathfrak{K}$=$\mathfrak{K}$, *unfolded cat-cs-simps*
]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-τ-components′*(*2–5*)

end

15.4.2 Natural transformation map

mk-VLambda *L-10-5-τ-components*(*1*)
|*vsv L-10-5-τ-NTMap-vsv*[*cat-Kan-cs-intros*]|
|*vdomain L-10-5-τ-NTMap-vdomain*[*cat-Kan-cs-simps*]|

lemma *L-10-5-τ-NTMap-app*[*cat-Kan-cs-simps*]:
assumes $bf = [0, b, f]_\circ$ **and** $bf \in_\circ c \downarrow_{CF} \mathfrak{K}(\lfloor Obj \rfloor)$
shows *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v $a(\lfloor NTMap \rfloor)(\lfloor bf \rfloor) = v(\lfloor NTMap \rfloor)(\lfloor b \rfloor)(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$
using *assms* **unfolding** *L-10-5-τ-components* **by** (*simp add*: *nat-omega-simps*)

15.4.3 *L-10-5-τ* is a cone

lemma *L-10-5-τ-is-cat-cone*[*cat-cs-intros*]:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
and v'-*def*: $v' =$ *ntcf-arrow* v
and v: $v :$
$Hom_{O.C\alpha}\mathfrak{C}(c,-) \circ_{CF} \mathfrak{K} \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
and a: $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
shows *L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ c v' $a : a <_{CF.cone} \mathfrak{T} \circ_{CF} c\ _O\!\bigsqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–

let *?H*-$\mathfrak{C}$ = $\langle \lambda c.\ Hom_{O.C\alpha}\mathfrak{C}(c,-) \rangle$

```
let ?H-𝔄 = ‹λa. Hom_{O.Cα}𝔄(a,−)›

interpret ℜ: is-functor α 𝔅 ℭ ℜ by (rule assms(1))
interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))

from assms(3) interpret cℜ: category α ‹c ↓_{CF} ℜ›
  by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)
from assms(3) interpret Πc: is-functor α ‹c ↓_{CF} ℜ› 𝔅 ‹c _O∏_{CF} ℜ›
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )
interpret υ: is-ntcf α 𝔅 ‹cat-Set α› ‹?H-ℭ c ∘_{CF} ℜ› ‹?H-𝔄 a ∘_{CF} 𝔗› υ
  by (rule υ)

show ?thesis
proof(intro is-cat-coneI is-ntcfI′)
  show vfsequence (L-10-5-τ 𝔗 ℜ c υ′ a) unfolding L-10-5-τ-def by simp
  show vcard (L-10-5-τ 𝔗 ℜ c υ′ a) = 5_ℕ
    unfolding L-10-5-τ-def by (simp add: nat-omega-simps)
  from a interpret cf-const:
    is-functor α ‹c ↓_{CF} ℜ› 𝔄 ‹cf-const (c ↓_{CF} ℜ) 𝔄 a›
    by (cs-concl cs-intro: cat-cs-intros)
  show L-10-5-τ 𝔗 ℜ c υ′ a⦇NTMap⦈⦇bf⦈ :
    cf-const (c ↓_{CF} ℜ) 𝔄 a⦇ObjMap⦈⦇bf⦈ ↦_𝔄 (𝔗 ∘_{CF} c _O∏_{CF} ℜ)⦇ObjMap⦈⦇bf⦈
    if bf ∈_∘ c ↓_{CF} ℜ⦇Obj⦈ for bf
  proof–
    from that assms(3) obtain b f
      where bf-def: bf = [0, b, f]_∘
        and b: b ∈_∘ 𝔅⦇Obj⦈
        and f: f : c ↦_ℭ ℜ⦇ObjMap⦈⦇b⦈
      by auto
    from υ.ntcf-NTMap-is-arr[OF b] a b assms(3) f have υ⦇NTMap⦈⦇b⦈ :
      Hom ℭ c (ℜ⦇ObjMap⦈⦇b⦈) ↦_{cat-Set α} Hom 𝔄 a (𝔗⦇ObjMap⦈⦇b⦈)
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
        )
    with that b f show L-10-5-τ 𝔗 ℜ c υ′ a⦇NTMap⦈⦇bf⦈ :
      cf-const (c ↓_{CF} ℜ) 𝔄 a⦇ObjMap⦈⦇bf⦈ ↦_𝔄 (𝔗 ∘_{CF} c _O∏_{CF} ℜ)⦇ObjMap⦈⦇bf⦈
      unfolding bf-def υ′-def
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-Kan-cs-simps
              cat-comma-cs-simps
              cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed

  show
```

```
    L-10-5-τ 𝔗 ℜ c υ' a(|NTMap|)(|B|) ∘_{A 𝔄} cf-const (c ↓_{CF} ℜ) 𝔄 a(|ArrMap|)(|F|) =
      (𝔗 ∘_{CF} c _O⨅_{CF} ℜ)(|ArrMap|)(|F|) ∘_{A 𝔄} L-10-5-τ 𝔗 ℜ c υ' a(|NTMap|)(|A|)
    if F : A ↦_{c ↓_{CF} ℜ} B for A B F
  proof–
    from ℜ.is-functor-axioms that assms(3) obtain a' f a'' f' g
      where F-def: F = [[0, a', f]∘, [0, a'', f']∘, [0, g]∘]∘
        and A-def: A = [0, a', f]∘
        and B-def: B = [0, a'', f']∘
        and g: g : a' ↦_{𝔅} a''
        and f: f : c ↦_{ℭ} ℜ(|ObjMap|)(|a'|)
        and f'-def: ℜ(|ArrMap|)(|g|) ∘_{A ℭ} f = f'
      by auto
    from υ.ntcf-Comp-commute[OF g] have
      (υ(|NTMap|)(|a''|) ∘_{A cat-Set α} (?H-ℭ c ∘_{CF} ℜ)(|ArrMap|)(|g|))(|ArrVal|)(|f|) =
        ((?H-𝔄 a ∘_{CF} 𝔗)(|ArrMap|)(|g|) ∘_{A cat-Set α} υ(|NTMap|)(|a'|))(|ArrVal|)(|f|)
      by simp
    from this a g f f' ℜ.HomCod.category-axioms 𝔗.HomCod.category-axioms
    have [cat-cs-simps]:
      υ(|NTMap|)(|a''|)(|ArrVal|)(ℜ(|ArrMap|)(|g|) ∘_{A ℭ} f) =
        𝔗(|ArrMap|)(|g|) ∘_{A 𝔄} υ(|NTMap|)(|a'|)(|ArrVal|)(|f|)
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-prod-cs-intros cat-op-intros
        )
    from that a g f f' ℜ.HomCod.category-axioms 𝔗.HomCod.category-axioms
    show ?thesis
      unfolding F-def A-def B-def υ'-def
      by
        (
          cs-concl
            cs-simp:
              f'-def[symmetric]
              cat-cs-simps
              cat-Kan-cs-simps
              cat-comma-cs-simps
              cat-FUNCT-cs-simps
              cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
        )
  qed

qed
  (
    use assms in
      ‹
        cs-concl
          cs-simp: cat-cs-simps cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-Kan-cs-intros a
      ›
  )+

qed

lemma L-10-5-τ-is-cat-cone'[cat-Kan-cs-intros]:
  assumes ℜ : 𝔅 ↦↦_{Cα} ℭ
```

and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
and $\upsilon' = ntcf\text{-}arrow\ \upsilon$
and $\mathfrak{F}' = \mathfrak{T} \circ_{CF} c\ {}_O\!\sqcap_{CF} \mathfrak{K}$
and $c\mathfrak{K} = c \downarrow_{CF} \mathfrak{K}$
and $\mathfrak{A}' = \mathfrak{A}$
and $\alpha' = \alpha$
and $\upsilon :$
$Hom_{O.C\alpha}\mathfrak{C}(c,-) \circ_{CF} \mathfrak{K} \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} :$
$\mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
and $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
shows *L-10-5-τ* $\mathfrak{T}\ \mathfrak{K}\ c\ \upsilon'\ a : a <_{CF.cone} \mathfrak{F}' : c\mathfrak{K} \mapsto\mapsto_{C_{\alpha'}} \mathfrak{A}'$
using *assms*(*1–4,9,10*) **unfolding** *assms*(*5–8*) **by** (*rule L-10-5-τ-is-cat-cone*)

15.5 Lemma X.5: *L-10-5-υ*

15.5.1 Definition and elementary properties

definition *L-10-5-υ* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *L-10-5-υ* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a =$
[
$(\lambda b \in_\circ \mathfrak{T}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ L\text{-}10\text{-}5\text{-}\upsilon\text{-}arrow\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a\ b)$,
$Hom_{O.C\alpha}\mathfrak{K}(\lfloor HomCod \rfloor)(c,-) \circ_{CF} \mathfrak{K}$,
$Hom_{O.C\alpha}\mathfrak{T}(\lfloor HomCod \rfloor)(a,-) \circ_{CF} \mathfrak{T}$,
$\mathfrak{T}(\lfloor HomDom \rfloor)$,
$cat\text{-}Set\ \alpha$
]$_\circ$

Components.

lemma *L-10-5-υ-components*:
shows *L-10-5-υ* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a(\lfloor NTMap \rfloor) =$
$(\lambda b \in_\circ \mathfrak{T}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ L\text{-}10\text{-}5\text{-}\upsilon\text{-}arrow\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a\ b)$
and *L-10-5-υ* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a(\lfloor NTDom \rfloor) = Hom_{O.C\alpha}\mathfrak{K}(\lfloor HomCod \rfloor)(c,-) \circ_{CF} \mathfrak{K}$
and *L-10-5-υ* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a(\lfloor NTCod \rfloor) = Hom_{O.C\alpha}\mathfrak{T}(\lfloor HomCod \rfloor)(a,-) \circ_{CF} \mathfrak{T}$
and *L-10-5-υ* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a(\lfloor NTDGDom \rfloor) = \mathfrak{T}(\lfloor HomDom \rfloor)$
and *L-10-5-υ* $\alpha\ \mathfrak{T}\ \mathfrak{K}\ c\ \tau\ a(\lfloor NTDGCod \rfloor) = cat\text{-}Set\ \alpha$
unfolding *L-10-5-υ-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{A}\ \mathfrak{K}\ \mathfrak{T}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)
interpretation $\mathfrak{T}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *L-10-5-υ-components′* = *L-10-5-υ-components*[
where $\mathfrak{T}$=$\mathfrak{T}$ **and** $\mathfrak{K}$=$\mathfrak{K}$, *unfolded cat-cs-simps*
]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-υ-components′*(*2–5*)

end

15.5.2 Natural transformation map

mk-VLambda *L-10-5-υ-components*(*1*)
|*vsv L-10-5-υ-NTMap-vsv*[*cat-Kan-cs-intros*]|

context
  fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{K}$ $\mathfrak{T}$
  assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{T}$: $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)
interpretation $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

mk-VLambda *L-10-5-υ-components′*(*1*)[*OF* $\mathfrak{K}$ $\mathfrak{T}$]
  |*vdomain L-10-5-υ-NTMap-vdomain*[*cat-Kan-cs-simps*]|
  |*app L-10-5-υ-NTMap-app*[*cat-Kan-cs-simps*]|

end

15.5.3 *L-10-5-υ* is a natural transformation

lemma *L-10-5-υ-is-ntcf*:
  assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
    and τ'-*def*: $\tau' = ntcf\text{-}arrow\ \tau$
    and τ: $\tau : a <_{CF.cone} \mathfrak{T} \circ_{CF} c\ {}_O\!\sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and a: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
  shows *L-10-5-υ* α $\mathfrak{T}$ $\mathfrak{K}$ c τ' a :
    $Hom_{O.C\alpha}\mathfrak{C}(c,-) \circ_{CF} \mathfrak{K} \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
    (**is** ‹*?L-10-5-υ* : *?H-*$\mathfrak{C}$ $c \circ_{CF} \mathfrak{K} \mapsto_{CF}$ *?H-*$\mathfrak{A}$ $a \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$›)
proof–

  interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*2*))

  interpret τ: *is-cat-cone* α a ‹$c \downarrow_{CF} \mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T} \circ_{CF} c\ {}_O\!\sqcap_{CF} \mathfrak{K}$› τ
    by (*rule assms*(*5*))

  from *assms*(*3*) **interpret** *c*$\mathfrak{K}$: *category* α ‹$c \downarrow_{CF} \mathfrak{K}$›
    by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
  from *assms*(*3*) **interpret** Πc: *is-functor* α ‹$c \downarrow_{CF} \mathfrak{K}$› $\mathfrak{B}$ ‹$c\ {}_O\!\sqcap_{CF} \mathfrak{K}$›
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-comma-cs-simps*
          cs-intro: *cat-cs-intros cat-comma-cs-intros*
)

  show *?L-10-5-υ* : *?H-*$\mathfrak{C}$ $c \circ_{CF} \mathfrak{K} \mapsto_{CF}$ *?H-*$\mathfrak{A}$ $a \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
  proof(*intro is-ntcfI′*)
    show *vfsequence ?L-10-5-υ* **unfolding** *L-10-5-υ-def* **by** *auto*
    show *vcard ?L-10-5-υ* = $5_{\mathbb{N}}$
      unfolding *L-10-5-υ-def* **by** (*simp add*: *nat-omega-simps*)
    show *?L-10-5-υ*(|*NTMap*|)(|b|) :
      (*?H-*$\mathfrak{C}$ $c \circ_{CF} \mathfrak{K}$)(|*ObjMap*|)(|$b$|) $\mapsto_{cat\text{-}Set\ \alpha}$ (*?H-*$\mathfrak{A}$ $a \circ_{CF} \mathfrak{T}$)(|*ObjMap*|)(|$b$|)
      if $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **for** b
    proof–
      from a *that assms*(*3*) **show** *?thesis*
        unfolding τ'-*def*
        by
          (

```
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-Kan-cs-simps
          cs-intro:
            cat-Kan-cs-intros
            cat-lim-cs-intros
            cat-cs-intros
            cat-op-intros
        )
    qed
    show
      ?L-10-5-υ(|NTMap|)(|b'|) ∘_A cat-Set α (?H-𝔈 c ∘_CF 𝔎)(|ArrMap|)(|f|) =
        (?H-𝔄 a ∘_CF 𝔗)(|ArrMap|)(|f|) ∘_A cat-Set α ?L-10-5-υ(|NTMap|)(|a'|)
      if f : a' ↦_𝔅 b' for a' b' f
    proof–
      from that a assms(3) show ?thesis
        by
          (
          cs-concl
            cs-simp: cat-cs-simps cat-Kan-cs-simps cat-op-simps τ'-def
            cs-intro: cat-lim-cs-intros cat-cs-intros
          )
    qed

  qed
    (
      use assms(3,6) in
        ‹
        cs-concl
          cs-simp: cat-cs-simps cat-Kan-cs-simps
          cs-intro: cat-cs-intros cat-Kan-cs-intros
        ›
    )+

qed

lemma L-10-5-υ-is-ntcf'[cat-Kan-cs-intros]:
  assumes 𝔎 : 𝔅 ↦↦_Cα 𝔈
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and c ∈_∘ 𝔈(|Obj|)
    and τ' = ntcf-arrow τ
    and 𝔉' = Hom_O.Cα𝔈(c,–) ∘_CF 𝔎
    and 𝔊' = Hom_O.Cα𝔄(a,–) ∘_CF 𝔗
    and 𝔅' = 𝔅
    and 𝔈' = cat-Set α
    and α' = α
    and τ : a <_CF.cone 𝔗 ∘_CF c _O⨅_CF 𝔎 : c ↓_CF 𝔎 ↦↦_Cα 𝔄
    and a ∈_∘ 𝔄(|Obj|)
  shows L-10-5-υ α 𝔗 𝔎 c τ' a : 𝔉' ↦_CF 𝔊' : 𝔅' ↦↦_C_α' 𝔈'
  using assms(1–4,10,11) unfolding assms(5–9) by (rule L-10-5-υ-is-ntcf)
```

15.6 Lemma X.5: *L-10-5-χ-arrow*

15.6.1 Definition and elementary properties

```
definition L-10-5-χ-arrow
  where L-10-5-χ-arrow α β 𝔗 𝔎 c a =
    [
      (λυ∈_∘L-10-5-N α β 𝔗 𝔎 c(|ObjMap|)(|a|). ntcf-arrow (L-10-5-τ 𝔗 𝔎 c υ a)),
```

L-10-5-N α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*ObjMap*|)(|*a*|),
cf-Cone α β ($\mathfrak{T}$ $\circ_{CF}$ *c* $_O\sqcap_{CF}$ $\mathfrak{K}$)(|*ObjMap*|)(|*a*|)
]$_\circ$

Components.

lemma *L-10-5-χ-arrow-components*:
shows *L-10-5-χ-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ *c a*(|*ArrVal*|) =
(λ$\upsilon\in_\circ$*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*ObjMap*|)(|*a*|). *ntcf-arrow* (*L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ *c* υ *a*))
and *L-10-5-χ-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ *c a*(|*ArrDom*|) = *L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*(|*ObjMap*|)(|*a*|)
and *L-10-5-χ-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ *c a*(|*ArrCod*|) =
cf-Cone α β ($\mathfrak{T}$ $\circ_{CF}$ *c* $_O\sqcap_{CF}$ $\mathfrak{K}$)(|*ObjMap*|)(|*a*|)
unfolding *L-10-5-χ-arrow-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

lemmas [*cat-Kan-cs-simps*] = *L-10-5-χ-arrow-components*(*2*,*3*)

15.6.2 Arrow value

mk-VLambda *L-10-5-χ-arrow-components*(*1*)
|*vsv L-10-5-χ-arrow-vsv*[*cat-Kan-cs-intros*]|
|*vdomain L-10-5-χ-arrow-vdomain*|
|*app L-10-5-χ-arrow-app*|

lemma *L-10-5-χ-arrow-vdomain′*[*cat-Kan-cs-simps*]:
assumes $\mathcal{Z}$ β
and α $\in_\circ$ β
and $\mathfrak{K}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{A}$
and *c* $\in_\circ$ $\mathfrak{C}$(|*Obj*|)
and *a* $\in_\circ$ $\mathfrak{A}$(|*Obj*|)
shows $\mathcal{D}_\circ$ (*L-10-5-χ-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ *c a*(|*ArrVal*|)) = *Hom*
(*cat-FUNCT* α $\mathfrak{B}$ (*cat-Set* α))
(*cf-map* (*Hom*$_{O.C\alpha}\mathfrak{C}$(*c*,−) $\circ_{CF}$ $\mathfrak{K}$))
(*cf-map* (*Hom*$_{O.C\alpha}\mathfrak{A}$(*a*,−) $\circ_{CF}$ $\mathfrak{T}$))
using *assms*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-Kan-cs-simps L-10-5-χ-arrow-vdomain*
cs-intro: *cat-cs-intros*
)

lemma *L-10-5-χ-arrow-app′*[*cat-Kan-cs-simps*]:
assumes $\mathcal{Z}$ β
and α $\in_\circ$ β
and $\mathfrak{K}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{A}$
and *c* $\in_\circ$ $\mathfrak{C}$(|*Obj*|)
and υ'-*def*: υ' = *ntcf-arrow* υ
and υ: υ :
Hom$_{O.C\alpha}\mathfrak{C}$(*c*,−) $\circ_{CF}$ $\mathfrak{K}$ $\mapsto_{CF}$ *Hom*$_{O.C\alpha}\mathfrak{A}$(*a*,−) $\circ_{CF}$ $\mathfrak{T}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ *cat-Set* α
and *a*: *a* $\in_\circ$ $\mathfrak{A}$(|*Obj*|)
shows
L-10-5-χ-arrow α β $\mathfrak{T}$ $\mathfrak{K}$ *c a*(|*ArrVal*|)(|υ'|) =
ntcf-arrow (*L-10-5-τ* $\mathfrak{T}$ $\mathfrak{K}$ *c* υ' *a*)
using *assms*
by
(

```
    cs-concl cs-shallow
      cs-simp: cat-cs-simps cat-Kan-cs-simps L-10-5-χ-arrow-app
      cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )

lemma υτa-def:
  assumes ℜ : 𝔅 ↦↦_Cα ℭ
    and 𝔗 : 𝔅 ↦↦_Cα 𝔄
    and c ∈_∘ ℭ⦇Obj⦈
    and υτa'-def: υτa' = ntcf-arrow υτa
    and υτa: υτa :
      Hom_O.Cα ℭ(c,−) ∘_CF ℜ ↦_CF Hom_O.Cα 𝔄(a,−) ∘_CF 𝔗 :
      𝔅 ↦↦_Cα cat-Set α
    and a: a ∈_∘ 𝔄⦇Obj⦈
  shows υτa = L-10-5-υ α 𝔗 ℜ c (ntcf-arrow (L-10-5-τ 𝔗 ℜ c υτa' a)) a
  (is ‹υτa = ?L-10-5-υ (ntcf-arrow ?L-10-5-τ) a›)
proof–

  interpret ℜ: is-functor α 𝔅 ℭ ℜ by (rule assms(1))
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))

  interpret υτa: is-ntcf
    α 𝔅 ‹cat-Set α› ‹Hom_O.Cα ℭ(c,−) ∘_CF ℜ› ‹Hom_O.Cα 𝔄(a,−) ∘_CF 𝔗› υτa
    by (rule υτa)

  show ?thesis
  proof(rule ntcf-eqI)
    show υτa :
      Hom_O.Cα ℭ(c,−) ∘_CF ℜ ↦_CF Hom_O.Cα 𝔄(a,−) ∘_CF 𝔗 : 𝔅 ↦↦_Cα cat-Set α
      by (rule υτa)
    from assms(1–3) a show
      ?L-10-5-υ (ntcf-arrow ?L-10-5-τ) a :
        Hom_O.Cα ℭ(c,−) ∘_CF ℜ ↦_CF Hom_O.Cα 𝔄(a,−) ∘_CF 𝔗 : 𝔅 ↦↦_Cα cat-Set α
      by
        (
          cs-concl
            cs-simp: cat-Kan-cs-simps υτa'-def
            cs-intro: cat-cs-intros cat-Kan-cs-intros
        )
    have dom-lhs: 𝒟_∘ (υτa⦇NTMap⦈) = 𝔅⦇Obj⦈
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    have dom-rhs: 𝒟_∘ (?L-10-5-υ (ntcf-arrow (?L-10-5-τ)) a⦇NTMap⦈) = 𝔅⦇Obj⦈
      by (cs-concl cs-shallow cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)
    show υτa⦇NTMap⦈ = ?L-10-5-υ (ntcf-arrow ?L-10-5-τ) a⦇NTMap⦈
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix b assume prems: b ∈_∘ 𝔅⦇Obj⦈
      from prems assms(3) a have lhs: υτa⦇NTMap⦈⦇b⦈ :
        Hom ℭ c (ℜ⦇ObjMap⦈⦇b⦈) ↦_cat-Set α Hom 𝔄 a (𝔗⦇ObjMap⦈⦇b⦈)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
          )
      then have dom-lhs: 𝒟_∘ (υτa⦇NTMap⦈⦇b⦈⦇ArrVal⦈) = Hom ℭ c (ℜ⦇ObjMap⦈⦇b⦈)
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      from prems assms(3) a have rhs:
        L-10-5-υ-arrow 𝔗 ℜ c (ntcf-arrow ?L-10-5-τ) a b :
          Hom ℭ c (ℜ⦇ObjMap⦈⦇b⦈) ↦_cat-Set α Hom 𝔄 a (𝔗⦇ObjMap⦈⦇b⦈)
```

```
    unfolding υτa'-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps
          cs-intro: cat-Kan-cs-intros cat-cs-intros
      )

  then have dom-rhs:
    𝒟∘ (L-10-5-υ-arrow 𝔗 ℜ c (ntcf-arrow ?L-10-5-τ) a b(|ArrVal|)) =
      Hom 𝔈 c (ℜ(|ObjMap|)(|b|))
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have [cat-cs-simps]:
    υτa(|NTMap|)(|b|) = L-10-5-υ-arrow 𝔗 ℜ c (ntcf-arrow ?L-10-5-τ) a b
  proof(rule arr-Set-eqI)
    from lhs show arr-Set-lhs: arr-Set α (υτa(|NTMap|)(|b|))
      by (auto dest: cat-Set-is-arrD(1))
    from rhs show arr-Set-rhs:
      arr-Set α (L-10-5-υ-arrow 𝔗 ℜ c (ntcf-arrow (?L-10-5-τ)) a b)
      by (auto dest: cat-Set-is-arrD(1))
    show υτa(|NTMap|)(|b|)(|ArrVal|) =
      L-10-5-υ-arrow 𝔗 ℜ c (ntcf-arrow ?L-10-5-τ) a b(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix f assume f : c ↦𝔈 ℜ(|ObjMap|)(|b|)
      with assms prems show
        υτa(|NTMap|)(|b|)(|ArrVal|)(|f|) =
          L-10-5-υ-arrow 𝔗 ℜ c (ntcf-arrow ?L-10-5-τ) a b(|ArrVal|)(|f|)
        unfolding υτa'-def
        by
          (
            cs-concl cs-shallow
              cs-simp:
                cat-Kan-cs-simps cat-FUNCT-cs-simps L-10-5-υ-arrow-ArrVal-app
              cs-intro: cat-cs-intros cat-comma-cs-intros
          )
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed (use lhs rhs in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

  from prems show
    υτa(|NTMap|)(|b|) = L-10-5-υ α 𝔗 ℜ c (ntcf-arrow ?L-10-5-τ) a(|NTMap|)(|b|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
      )

qed (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros V-cs-intros)+

qed simp-all

qed
```

15.7 Lemma X.5: *L-10-5-χ'-arrow*

15.7.1 Definition and elementary properties

```
definition L-10-5-χ'-arrow :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where L-10-5-χ'-arrow α β 𝔗 ℜ c a =
```

```
  [
    (
      λτ∈₀cf-Cone α β (𝔗 ∘_CF c _O⨅_CF 𝔎)(⦇ObjMap⦈)(⦇a⦈).
        ntcf-arrow (L-10-5-υ α 𝔗 𝔎 c τ a)
    ),
    cf-Cone α β (𝔗 ∘_CF c _O⨅_CF 𝔎)(⦇ObjMap⦈)(⦇a⦈),
    L-10-5-N α β 𝔗 𝔎 c(⦇ObjMap⦈)(⦇a⦈)
  ]₀
```

Components.

```
lemma L-10-5-χ'-arrow-components:
  shows L-10-5-χ'-arrow α β 𝔗 𝔎 c a(⦇ArrVal⦈) =
    (
      λτ∈₀cf-Cone α β (𝔗 ∘_CF c _O⨅_CF 𝔎)(⦇ObjMap⦈)(⦇a⦈).
        ntcf-arrow (L-10-5-υ α 𝔗 𝔎 c τ a)
    )
    and [cat-Kan-cs-simps]: L-10-5-χ'-arrow α β 𝔗 𝔎 c a(⦇ArrDom⦈) =
      cf-Cone α β (𝔗 ∘_CF c _O⨅_CF 𝔎)(⦇ObjMap⦈)(⦇a⦈)
    and [cat-Kan-cs-simps]: L-10-5-χ'-arrow α β 𝔗 𝔎 c a(⦇ArrCod⦈) =
       L-10-5-N α β 𝔗 𝔎 c(⦇ObjMap⦈)(⦇a⦈)
  unfolding L-10-5-χ'-arrow-def arr-field-simps by (simp-all add: nat-omega-simps)
```

15.7.2 Arrow value

```
mk-VLambda L-10-5-χ'-arrow-components(1)
  |vsv L-10-5-χ'-arrow-ArrVal-vsv[cat-Kan-cs-intros]|
  |vdomain L-10-5-χ'-arrow-ArrVal-vdomain|
  |app L-10-5-χ'-arrow-ArrVal-app|

lemma L-10-5-χ'-arrow-ArrVal-vdomain'[cat-Kan-cs-simps]:
  assumes ℤ β
    and α ∈₀ β
    and τ: τ : a <_CF.cone 𝔗 ∘_CF c _O⨅_CF 𝔎 : c ↓_CF 𝔎 ↦↦_Cα 𝔄
    and a: a ∈₀ 𝔄(⦇Obj⦈)
  shows 𝒟₀ (L-10-5-χ'-arrow α β 𝔗 𝔎 c a(⦇ArrVal⦈)) = Hom
    (cat-FUNCT α (c ↓_CF 𝔎) 𝔄)
    (cf-map (cf-const (c ↓_CF 𝔎) 𝔄 a))
    (cf-map (𝔗 ∘_CF c _O⨅_CF 𝔎))
proof–
  interpret β: ℤ β by (rule assms(1))
  interpret τ: is-cat-cone α a ‹c ↓_CF 𝔎› 𝔄 ‹𝔗 ∘_CF c _O⨅_CF 𝔎› τ
    by (rule assms(3))
  from assms(1,2,4) show ?thesis
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps L-10-5-χ'-arrow-ArrVal-vdomain
          cs-intro: cat-cs-intros
      )
qed

lemma L-10-5-χ'-arrow-ArrVal-app'[cat-cs-simps]:
  assumes ℤ β
    and α ∈₀ β
    and τ'-def: τ' = ntcf-arrow τ
    and τ: τ : a <_CF.cone 𝔗 ∘_CF c _O⨅_CF 𝔎 : c ↓_CF 𝔎 ↦↦_Cα 𝔄
    and a: a ∈₀ 𝔄(⦇Obj⦈)
  shows L-10-5-χ'-arrow α β 𝔗 𝔎 c a(⦇ArrVal⦈)(⦇τ'⦈) =
```

ntcf-arrow (*L-10-5-υ* α $\mathfrak{T}$ $\mathfrak{K}$ c τ' a)
proof–
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
interpret τ: *is-cat-cone* α a ‹c $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{A}$ ‹$\mathfrak{T}$ $\circ_{CF}$ c $_O\bigsqcap_{CF}$ $\mathfrak{K}$› τ
by (*rule assms*(*4*))
from *assms*(*2*,*5*) **have** $\tau' \in_\circ$ *cf-Cone* α β ($\mathfrak{T}$ $\circ_{CF}$ c $_O\bigsqcap_{CF}$ $\mathfrak{K}$)(⦇*ObjMap*⦈)(⦇a⦈)
unfolding τ'-*def*
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-FUNCT-cs-intros cat-cs-intros*
)
then show
L-10-5-χ'*-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ c a(⦇*ArrVal*⦈)(⦇τ'⦈) =
ntcf-arrow (*L-10-5-υ* α $\mathfrak{T}$ $\mathfrak{K}$ c τ' a)
unfolding *L-10-5-*χ'*-arrow-components* **by** *auto*
qed

15.7.3 *L-10-5-*χ'*-arrow* is an isomorphism in the category *Set*

lemma *L-10-5-*χ'*-arrow-is-iso-arr*:
assumes $\mathcal{Z}$ β
and $\alpha \in_\circ \beta$
and $\mathfrak{K}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{A}$
and $c \in_\circ \mathfrak{C}$(⦇*Obj*⦈)
and $a \in_\circ \mathfrak{A}$(⦇*Obj*⦈)
shows *L-10-5-*χ'*-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ c a :
cf-Cone α β ($\mathfrak{T}$ $\circ_{CF}$ c $_O\bigsqcap_{CF}$ $\mathfrak{K}$)(⦇*ObjMap*⦈)(⦇a⦈) $\mapsto_{iso}$ *cat-Set* β
L-10-5-N α β $\mathfrak{T}$ $\mathfrak{K}$ c(⦇*ObjMap*⦈)(⦇a⦈)
(
is
‹
?L-10-5-χ'*-arrow* :
cf-Cone α β ($\mathfrak{T}$ $\circ_{CF}$ c $_O\bigsqcap_{CF}$ $\mathfrak{K}$)(⦇*ObjMap*⦈)(⦇a⦈) $\mapsto_{iso}$ *cat-Set* β
?L-10-5-N(⦇*ObjMap*⦈)(⦇a⦈)
›
)
proof–

let *?FUNCT* = ‹$\lambda\mathfrak{A}$. *cat-FUNCT* α $\mathfrak{A}$ (*cat-Set* α)›
let *?c*$\mathfrak{K}$-$\mathfrak{A}$ = ‹*cat-FUNCT* α (c $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$›
let *?H*-$\mathfrak{C}$ = ‹λc. $Hom_{O.C\alpha}\mathfrak{C}(c,-)$›
let *?H*-$\mathfrak{A}$ = ‹λc. $Hom_{O.C\alpha}\mathfrak{A}(a,-)$›

from *assms*(*1*,*2*) **interpret** β: $\mathcal{Z}$ β **by** *simp*

interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*4*))

from $\mathfrak{K}$.*vempty-is-zet assms* **interpret** *c*$\mathfrak{K}$: *category* α ‹c $\downarrow_{CF}$ $\mathfrak{K}$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
from *assms*(*2*,*6*) **interpret** *c*$\mathfrak{K}$-$\mathfrak{A}$: *category* β *?c*$\mathfrak{K}$-$\mathfrak{A}$
by
(
cs-concl **cs-intro**:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros

```
  )
from ℜ.vempty-is-zet assms interpret Πc:
  is-functor α ‹c ↓CF ℜ› 𝔅 ‹c OCF⨅ CF ℜ›
  by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)

from assms(2) interpret FUNCT-𝔄: tiny-category β ‹?FUNCT 𝔄›
  by (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
from assms(2) interpret FUNCT-𝔅: tiny-category β ‹?FUNCT 𝔅›
  by (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
from assms(2) interpret FUNCT-ℭ: tiny-category β ‹?FUNCT ℭ›
  by (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)

have 𝔗Π: 𝔗 ∘CF c OCF⨅ CF ℜ : c ↓CF ℜ ↦↦Cα 𝔄
  by (cs-concl cs-intro: cat-cs-intros)

from assms(5,6) have [cat-cs-simps]:
  cf-of-cf-map (c ↓CF ℜ) 𝔄 (cf-map (cf-const (c ↓CF ℜ) 𝔄 a)) =
    cf-const (c ↓CF ℜ) 𝔄 a
  cf-of-cf-map (c ↓CF ℜ) 𝔄 (cf-map (𝔗 ∘CF c OCF⨅ CF ℜ)) = 𝔗 ∘CF c OCF⨅ CF ℜ
  cf-of-cf-map 𝔅 (cat-Set α) (cf-map (HomO.Cα ℭ(c,−) ∘CF ℜ)) =
    HomO.Cα ℭ(c,−) ∘CF ℜ
  cf-of-cf-map 𝔅 (cat-Set α) (cf-map (HomO.Cα 𝔄(a,−) ∘CF 𝔗)) =
    HomO.Cα 𝔄(a,−) ∘CF 𝔗
  by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)+

note cf-Cone-ObjMap-app = is-functor.cf-Cone-ObjMap-app[OF 𝔗Π assms(1,2,6)]

show ?thesis
proof
  (
    intro cat-Set-is-iso-arrI cat-Set-is-arrI arr-SetI,
    unfold L-10-5-χ′-arrow-components(3) cf-Cone-ObjMap-app
  )
  show vfsequence ?L-10-5-χ′-arrow
    unfolding L-10-5-χ′-arrow-def by auto
  show χ′-arrow-ArrVal-vsv: vsv (?L-10-5-χ′-arrow(|ArrVal|))
    unfolding L-10-5-χ′-arrow-components by auto
  show vcard ?L-10-5-χ′-arrow = 3ℕ
    unfolding L-10-5-χ′-arrow-def by (simp add: nat-omega-simps)
  show [cat-cs-simps]:
    𝒟∘ (?L-10-5-χ′-arrow(|ArrVal|)) = ?L-10-5-χ′-arrow(|ArrDom|)
    unfolding L-10-5-χ′-arrow-components by simp
  show vrange-χ′-arrow-vsubset-N′′:
    ℛ∘ (?L-10-5-χ′-arrow(|ArrVal|)) ⊆∘ ?L-10-5-N(|ObjMap|)(|a|)
    unfolding L-10-5-χ′-arrow-components
  proof(rule vrange-VLambda-vsubset)
    fix τ assume prems: τ ∈∘ cf-Cone α β (𝔗 ∘CF c OCF⨅ CF ℜ)(|ObjMap|)(|a|)
    from this assms cℜ-𝔄.category-axioms have τ-is-arr:
      τ : cf-map (cf-const (c ↓CF ℜ) 𝔄 a) ↦?cℜ-𝔄 cf-map (𝔗 ∘CF c OCF⨅ CF ℜ)
      by
        (
          cs-prems
            cs-simp: cat-cs-simps cat-Kan-cs-simps cat-FUNCT-components(1)
            cs-intro: cat-cs-intros
        )
    note τ = cat-FUNCT-is-arrD(1,2)[OF τ-is-arr, unfolded cat-cs-simps]
    have cf-of-cf-map (c ↓CF ℜ) 𝔄 (cf-map (𝔗 ∘CF c OCF⨅ CF ℜ)) = 𝔗 ∘CF c OCF⨅ CF ℜ
      by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)
```

```
  from prems assms τ(1) show
    ntcf-arrow (L-10-5-υ α 𝔗 ℜ c τ a) ∈∘ ?L-10-5-N(⁅ObjMap⁆)(⁅a⁆)
    by (subst τ(2))
      (
        cs-concl
          cs-simp: cat-cs-simps cat-Kan-cs-simps
          cs-intro:
            is-cat-coneI cat-cs-intros cat-Kan-cs-intros cat-FUNCT-cs-intros
      )
qed

show ℛ∘ (?L-10-5-χ'-arrow(⁅ArrVal⁆)) = ?L-10-5-N(⁅ObjMap⁆)(⁅a⁆)
proof
  (
    intro vsubset-antisym[OF vrange-χ'-arrow-vsubset-N''],
    intro vsubsetI
  )

  fix υτa assume υτa ∈∘ ?L-10-5-N(⁅ObjMap⁆)(⁅a⁆)
  from this assms have υτa:
    υτa : cf-map (?H-𝔈 c ∘CF ℜ) ↦?FUNCT 𝔅 cf-map (?H-𝔄 a ∘CF 𝔗)
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
      )
  note υτa = cat-FUNCT-is-arrD[OF this, unfolded cat-cs-simps]
  interpret τ:
    is-cat-cone α a ‹c ↓CF ℜ› 𝔄 ‹𝔗 ∘CF c ∘CF ℜ› ‹L-10-5-τ 𝔗 ℜ c υτa a›
    by (rule L-10-5-τ-is-cat-cone[OF assms(3,4,5) υτa(2,1) assms(6)])

  show υτa ∈∘ ℛ∘ (?L-10-5-χ'-arrow(⁅ArrVal⁆))
  proof(rule vsv.vsv-vimageI2')
    show vsv (?L-10-5-χ'-arrow(⁅ArrVal⁆)) by (rule χ'-arrow-ArrVal-vsv)
    from τ.is-cat-cone-axioms assms show
      ntcf-arrow (L-10-5-τ 𝔗 ℜ c υτa a) ∈∘ 𝒟∘ (?L-10-5-χ'-arrow(⁅ArrVal⁆))
      by
        (
          cs-concl
            cs-simp: cat-Kan-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
    from assms υτa(1,2) show
      υτa = ?L-10-5-χ'-arrow(⁅ArrVal⁆)(⁅ntcf-arrow (L-10-5-τ 𝔗 ℜ c υτa a)⁆)
      by
        (
          subst υτa(2),
          cs-concl-step υτa-def[OF assms(3,4,5) υτa(2,1) assms(6)]
        )
        (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed
qed

from assms show ?L-10-5-χ'-arrow(⁅ArrDom⁆) ∈∘ Vset β
  by
    (
      cs-concl
        cs-simp: cat-Kan-cs-simps cat-FUNCT-components(1) cf-Cone-ObjMap-app
```

```
      cs-intro: cat-cs-intros cat-FUNCT-cs-intros cℜ-𝔄.cat-Hom-in-Vset
    )
with assms(2) have ?L-10-5-χ'-arrow(|ArrDom|) ∈∘ Vset β
  by (meson Vset-in-mono Vset-trans)
from assms show ?L-10-5-N(|ObjMap|)(|a|) ∈∘ Vset β
  by
    (
      cs-concl
        cs-simp: cat-cs-simps cat-Kan-cs-simps cat-FUNCT-cs-simps
        cs-intro: cat-cs-intros FUNCT-𝔅.cat-Hom-in-Vset cat-FUNCT-cs-intros
    )
show dom-χ'-arrow: 𝒟∘ (?L-10-5-χ'-arrow(|ArrVal|)) =
  Hom ?cℜ-𝔄 (cf-map (cf-const (c ↓CF ℜ) 𝔄 a)) (cf-map (𝔗 ∘CF c ∘CF ℜ))
  unfolding L-10-5-χ'-arrow-components cf-Cone-ObjMap-app by simp
show ?L-10-5-χ'-arrow(|ArrDom|) =
  Hom ?cℜ-𝔄 (cf-map (cf-const (c ↓CF ℜ) 𝔄 a)) (cf-map (𝔗 ∘CF c ∘CF ℜ))
  unfolding L-10-5-χ'-arrow-components cf-Cone-ObjMap-app by simp
show v11 (?L-10-5-χ'-arrow(|ArrVal|))
proof(rule vsv.vsv-valeq-v11I, unfold dom-χ'-arrow in-Hom-iff)
  fix τ' τ'' assume prems:
    τ' : cf-map (cf-const (c ↓CF ℜ) 𝔄 a) ↦?cℜ-𝔄 cf-map (𝔗 ∘CF c ∘CF ℜ)
    τ'' : cf-map (cf-const (c ↓CF ℜ) 𝔄 a) ↦?cℜ-𝔄 cf-map (𝔗 ∘CF c ∘CF ℜ)
    ?L-10-5-χ'-arrow(|ArrVal|)(|τ'|) = ?L-10-5-χ'-arrow(|ArrVal|)(|τ''|)
  note τ' = cat-FUNCT-is-arrD[OF prems(1), unfolded cat-cs-simps]
    and τ'' = cat-FUNCT-is-arrD[OF prems(2), unfolded cat-cs-simps]
  interpret τ': is-cat-cone
    α a ‹c ↓CF ℜ› 𝔄 ‹𝔗 ∘CF c ∘CF ℜ› ‹ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ'›
    by (rule is-cat-coneI[OF τ'(1) assms(6)])
  interpret τ'': is-cat-cone
    α a ‹c ↓CF ℜ› 𝔄 ‹𝔗 ∘CF c ∘CF ℜ› ‹ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ''›
    by (rule is-cat-coneI[OF τ''(1) assms(6)])
  have τ'τ': ntcf-arrow (ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ') = τ'
    by (subst (2) τ'(2)) (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)
  have τ''τ'': ntcf-arrow (ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ'') = τ''
    by (subst (2) τ''(2)) (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)
  from prems(3) τ'(1) τ''(1) assms have
    L-10-5-υ α 𝔗 ℜ c τ' a = L-10-5-υ α 𝔗 ℜ c τ'' a
    by (subst (asm) τ'(2), use nothing in ‹subst (asm) τ''(2)›)
      (
        cs-prems cs-shallow
          cs-simp: τ'τ' τ''τ'' cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-lim-cs-intros cat-Kan-cs-intros cat-cs-intros
      )
  from this have υτ'a-υτ''a:
    L-10-5-υ α 𝔗 ℜ c τ' a(|NTMap|)(|b|)(|ArrVal|)(|f|) =
      L-10-5-υ α 𝔗 ℜ c τ'' a(|NTMap|)(|b|)(|ArrVal|)(|f|)
    if b ∈∘ 𝔅(|Obj|) and f : c ↦ℭ (ℜ(|ObjMap|)(|b|)) for b f
    by simp
  have [cat-cs-simps]: τ'(|NTMap|)(|0, b, f|)• = τ''(|NTMap|)(|0, b, f|)•
    if b ∈∘ 𝔅(|Obj|) and f : c ↦ℭ (ℜ(|ObjMap|)(|b|)) for b f
    using υτ'a-υτ''a[OF that] that
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-Kan-cs-simps L-10-5-υ-arrow-ArrVal-app
          cs-intro: cat-cs-intros
      )
  have
```

```
      ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ' =
        ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ''
    proof(rule ntcf-eqI)
      show ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ' :
        cf-const (c ↓CF ℜ) 𝔄 a ↦CF 𝔗 ∘CF c ∘CF ⨅CF ℜ : c ↓CF ℜ ↦↦→Cα 𝔄
        by (rule τ'.is-ntcf-axioms)
      then have dom-lhs:
        𝒟∘ (ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ'(‖NTMap‖)) = c ↓CF ℜ(‖Obj‖)
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      show ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ'' :
        cf-const (c ↓CF ℜ) 𝔄 a ↦CF 𝔗 ∘CF c ∘CF ⨅CF ℜ : c ↓CF ℜ ↦↦→Cα 𝔄
        by (rule τ''.is-ntcf-axioms)
      then have dom-rhs:
        𝒟∘ (ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ''(‖NTMap‖)) = c ↓CF ℜ(‖Obj‖)
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      show
        ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ'(‖NTMap‖) =
          ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ''(‖NTMap‖)
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
        fix A assume A ∈∘ c ↓CF ℜ(‖Obj‖)
        with assms(5) obtain b f
          where A-def: A = [0, b, f]∘
            and b: b ∈∘ 𝔅(‖Obj‖)
            and f: f : c ↦ℭ ℜ(‖ObjMap‖)(‖b‖)
          by auto
        from b f show
          ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ'(‖NTMap‖)(‖A‖) =
            ntcf-of-ntcf-arrow (c ↓CF ℜ) 𝔄 τ''(‖NTMap‖)(‖A‖)
          unfolding A-def
          by (cs-concl cs-simp: cat-cs-simps cat-map-extra-cs-simps)
      qed (cs-concl cs-shallow cs-intro: V-cs-intros)+
    qed simp-all
    then show τ' = τ''
    proof(rule inj-onD[OF bij-betw-imp-inj-on[OF bij-betw-ntcf-of-ntcf-arrow]])
      show τ' ∈∘ ntcf-arrows α (c ↓CF ℜ) 𝔄
        by (subst τ'(2))
          (
            cs-concl cs-intro:
              cat-lim-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )
      show τ'' ∈∘ ntcf-arrows α (c ↓CF ℜ) 𝔄
        by (subst τ''(2))
          (
            cs-concl cs-intro:
              cat-lim-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )
    qed
  qed (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)

 qed auto

qed

lemma L-10-5-χ'-arrow-is-iso-arr'[cat-Kan-cs-intros]:
  assumes 𝒵 β
    and α ∈∘ β
    and ℜ : 𝔅 ↦↦→Cα ℭ
    and 𝔗 : 𝔅 ↦↦→Cα 𝔄
```

and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $A = \textit{cf-Cone}\ \alpha\ \beta\ (\mathfrak{T} \circ_{CF} c\ {}_O\!\bigsqcap_{CF} \mathfrak{K})(\!|ObjMap|\!)(\!|a|\!)$
and $B = \textit{L-10-5-N}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c(\!|ObjMap|\!)(\!|a|\!)$
and $\mathfrak{C}' = \textit{cat-Set}\ \beta$
shows $\textit{L-10-5-}\chi'\textit{-arrow}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c\ a : A \mapsto_{iso\mathfrak{C}'} B$
using *assms*(*1–6*)
unfolding *assms*(*7–9*)
by (*rule L-10-5-*χ'*-arrow-is-iso-arr*)

lemma *L-10-5-*χ'*-arrow-is-arr*:
assumes $\mathcal{Z}\ \beta$
and $\alpha \in_\circ \beta$
and $\mathfrak{K} : \mathfrak{B} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\textit{L-10-5-}\chi'\textit{-arrow}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c\ a :$
$\textit{cf-Cone}\ \alpha\ \beta\ (\mathfrak{T} \circ_{CF} c\ {}_O\!\bigsqcap_{CF} \mathfrak{K})(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\textit{cat-Set}\ \beta}$
$\textit{L-10-5-N}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c(\!|ObjMap|\!)(\!|a|\!)$
by
(
rule cat-Set-is-iso-arrD(*1*)[
OF L-10-5-χ'*-arrow-is-iso-arr*[*OF assms*(*1–6*)]
]
)

lemma *L-10-5-*χ'*-arrow-is-arr'*[*cat-Kan-cs-intros*]:
assumes $\mathcal{Z}\ \beta$
and $\alpha \in_\circ \beta$
and $\mathfrak{K} : \mathfrak{B} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $A = \textit{cf-Cone}\ \alpha\ \beta\ (\mathfrak{T} \circ_{CF} c\ {}_O\!\bigsqcap_{CF} \mathfrak{K})(\!|ObjMap|\!)(\!|a|\!)$
and $B = \textit{L-10-5-N}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c(\!|ObjMap|\!)(\!|a|\!)$
and $\mathfrak{C}' = \textit{cat-Set}\ \beta$
shows $\textit{L-10-5-}\chi'\textit{-arrow}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c\ a : A \mapsto_{\mathfrak{C}'} B$
using *assms*(*1–6*) **unfolding** *assms*(*7–9*) **by** (*rule L-10-5-*χ'*-arrow-is-arr*)

15.8 Lemma X.5: *L-10-5-*χ

15.8.1 Definition and elementary properties

definition $\textit{L-10-5-}\chi :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where $\textit{L-10-5-}\chi\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c =$
[
$(\lambda a \in_\circ \mathfrak{T}(\!|HomCod|\!)(\!|Obj|\!).\ \textit{L-10-5-}\chi'\textit{-arrow}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c\ a)$,
$\textit{cf-Cone}\ \alpha\ \beta\ (\mathfrak{T} \circ_{CF} c\ {}_O\!\bigsqcap_{CF} \mathfrak{K})$,
$\textit{L-10-5-N}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c$,
$\textit{op-cat}\ (\mathfrak{T}(\!|HomCod|\!))$,
$\textit{cat-Set}\ \beta$
]$_\circ$

Components.

lemma *L-10-5-*χ*-components*:
shows $\textit{L-10-5-}\chi\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c(\!|NTMap|\!) =$
$(\lambda a \in_\circ \mathfrak{T}(\!|HomCod|\!)(\!|Obj|\!).\ \textit{L-10-5-}\chi'\textit{-arrow}\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c\ a)$

and [*cat-Kan-cs-simps*]:
L-10-5-χ α β 𝔗 ℜ c(|*NTDom*|) = *cf-Cone α β* (*𝔗* ∘$_{CF}$ *c* $_O\!\bigsqcap_{CF}$ *ℜ*)
and [*cat-Kan-cs-simps*]:
L-10-5-χ α β 𝔗 ℜ c(|*NTCod*|) = *L-10-5-N α β 𝔗 ℜ c*
and *L-10-5-χ α β 𝔗 ℜ c*(|*NTDGDom*|) = *op-cat* (*𝔗*(|*HomCod*|))
and [*cat-Kan-cs-simps*]: *L-10-5-χ α β 𝔗 ℜ c*(|*NTDGCod*|) = *cat-Set β*
unfolding *L-10-5-χ-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes *α 𝔄 𝔅 𝔗*
assumes *𝔗*: *𝔗* : *𝔅* ↦↦$_{C α}$ *𝔄*
begin

interpretation *is-functor α 𝔅 𝔄 𝔗* **by** (*rule 𝔗*)

lemmas *L-10-5-χ-components′* =
L-10-5-χ-components[**where** *𝔗*=*𝔗*, *unfolded cat-cs-simps*]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-χ-components′*(*4*)

end

15.8.2 Natural transformation map

mk-VLambda *L-10-5-χ-components*(*1*)
|*vsv L-10-5-χ-NTMap-vsv*[*cat-Kan-cs-intros*]|

context
fixes *α 𝔄 𝔅 𝔗*
assumes *𝔗*: *𝔗* : *𝔅* ↦↦$_{C α}$ *𝔄*
begin

interpretation *is-functor α 𝔅 𝔄 𝔗* **by** (*rule 𝔗*)

mk-VLambda *L-10-5-χ-components*(*1*)[**where** *𝔗*=*𝔗*, *unfolded cat-cs-simps*]
|*vdomain L-10-5-χ-NTMap-vdomain*[*cat-Kan-cs-simps*]|
|*app L-10-5-χ-NTMap-app*[*cat-Kan-cs-simps*]|

end

15.8.3 *L-10-5-χ* is a natural isomorphism

lemma *L-10-5-χ-is-iso-ntcf*:
— See lemma on page 245 in [9].
assumes *𝒵 β*
and *α* ∈$_\circ$ *β*
and *ℜ* : *𝔅* ↦↦$_{C α}$ *ℭ*
and *𝔗* : *𝔅* ↦↦$_{C α}$ *𝔄*
and *c* ∈$_\circ$ *ℭ*(|*Obj*|)
shows *L-10-5-χ α β 𝔗 ℜ c* :
cf-Cone α β (*𝔗* ∘$_{CF}$ *c* $_O\!\bigsqcap_{CF}$ *ℜ*) ↦$_{CF.iso}$ *L-10-5-N α β 𝔗 ℜ c* :
op-cat 𝔄 ↦↦$_{C β}$ *cat-Set β*
(**is** ‹*?L-10-5-χ* : *?cf-Cone* ↦$_{CF.iso}$ *?L-10-5-N* : *op-cat 𝔄* ↦↦$_{C β}$ *cat-Set β*›)
proof–

let *?FUNCT* = ‹*λ𝔄*. *cat-FUNCT α 𝔄* (*cat-Set α*)›
let *?cℜ-𝔄* = ‹*cat-FUNCT α* (*c* ↓$_{CF}$ *ℜ*) *𝔄*›
let *?ntcf-cℜ-𝔄* = ‹*ntcf-const* (*c* ↓$_{CF}$ *ℜ*) *𝔄*›

let *?𝔗-c𝔎* = ‹$\mathfrak{T}$ $\circ_{CF}$ *c* $_O\prod_{CF}$ $\mathfrak{K}$›
let *?H-𝔈* = ‹$\lambda c.\ Hom_{O.C\alpha}\mathfrak{C}(c,-)$›
let *?H-𝔄* = ‹$\lambda a.\ Hom_{O.C\alpha}\mathfrak{A}(a,-)$›
let *?L-10-5-χ′-arrow* = ‹*L-10-5-χ′-arrow* α β $\mathfrak{T}$ $\mathfrak{K}$ *c*›
let *?cf-c𝔎-𝔄* = ‹*cf-const* (*c* $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$›
let *?L-10-5-υ* = ‹*L-10-5-υ* α $\mathfrak{T}$ $\mathfrak{K}$ *c*›
let *?L-10-5-υ-arrow* = ‹*L-10-5-υ-arrow* $\mathfrak{T}$ $\mathfrak{K}$ *c*›

interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))

interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*4*))

from $\mathfrak{K}$.*vempty-is-zet assms*(*5*) **interpret** *c𝔎*: *category* α ‹*c* $\downarrow_{CF}$ $\mathfrak{K}$›
  by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
from *assms*(*1*,*2*,*5*) **interpret** *c𝔎-𝔄*: *category* β *?c𝔎-𝔄*
  by
    (
      cs-concl **cs-intro**:
        cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)
interpret β*-c𝔎-𝔄*: *category* β *?c𝔎-𝔄*
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros assms*(*2*))+
from *assms*(*2*,*5*) **interpret** Δ: *is-functor* β $\mathfrak{A}$ *?c𝔎-𝔄* ‹Δ_{CF} α (*c* $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$›
  by (*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros*)+
from $\mathfrak{K}$.*vempty-is-zet assms*(*5*) **interpret** Πc:
  is-functor α ‹*c* $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{B}$ ‹*c* $_O\prod_{CF}$ $\mathfrak{K}$›
  by
    (
      cs-concl **cs-shallow**
        cs-simp: *cat-comma-cs-simps*
        cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
interpret $\beta\Pi c$: *is-tiny-functor* β ‹*c* $\downarrow_{CF}$ $\mathfrak{K}$› $\mathfrak{B}$ ‹*c* $_O\prod_{CF}$ $\mathfrak{K}$›
  by (*rule* Πc.*cf-is-tiny-functor-if-ge-Limit*[*OF assms*(*1*,*2*)])

interpret *E*: *is-functor* β ‹*?FUNCT* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$› ‹*cat-Set* β› ‹*cf-eval* α β $\mathfrak{C}$›
  by (*rule* $\mathfrak{K}$.*HomCod.cat-cf-eval-is-functor*[*OF assms*(*1*,*2*)])

from *assms*(*2*) **interpret** *FUNCT-𝔄*: *tiny-category* β ‹*?FUNCT* $\mathfrak{A}$›
  by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *assms*(*2*) **interpret** *FUNCT-𝔅*: *tiny-category* β ‹*?FUNCT* $\mathfrak{B}$›
  by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *assms*(*2*) **interpret** *FUNCT-𝔈*: *tiny-category* β ‹*?FUNCT* $\mathfrak{C}$›
  by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)

interpret $\beta\mathfrak{A}$: *tiny-category* β $\mathfrak{A}$
  by (*rule category.cat-tiny-category-if-ge-Limit*)
    (*use assms*(*2*) **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{B}$: *tiny-category* β $\mathfrak{B}$
  by (*rule category.cat-tiny-category-if-ge-Limit*)
    (*use assms*(*2*) **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{C}$: *tiny-category* β $\mathfrak{C}$
  by (*rule category.cat-tiny-category-if-ge-Limit*)
    (*use assms*(*2*) **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+

interpret $\beta\mathfrak{K}$: *is-tiny-functor* β $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$
  by (*rule is-functor.cf-is-tiny-functor-if-ge-Limit*)

```
    (use assms(2) in ‹cs-concl cs-intro: cat-cs-intros›)+
interpret βT: is-tiny-functor β 𝔅 𝔄 T
  by (rule is-functor.cf-is-tiny-functor-if-ge-Limit)
    (use assms(2) in ‹cs-concl cs-intro: cat-cs-intros›)+

interpret cat-Set-αβ: subcategory β ‹cat-Set α› ‹cat-Set β›
  by (rule ℜ.subcategory-cat-Set-cat-Set[OF assms(1,2)])

show ?thesis
proof(intro is-iso-ntcfI is-ntcfI', unfold cat-op-simps)

  show vfsequence (?L-10-5-χ) unfolding L-10-5-χ-def by auto
  show vcard (?L-10-5-χ) = 5ℕ
    unfolding L-10-5-χ-def by (simp add: nat-omega-simps)
  from assms(2) show ?cf-Cone : op-cat 𝔄 ↦↦C β cat-Set β
    by (intro is-functor.tm-cf-cf-Cone-is-functor-if-ge-Limit)
      (cs-concl cs-intro: cat-cs-intros)+

  from assms show ?L-10-5-N : op-cat 𝔄 ↦↦C β cat-Set β
    by (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)
  show ?L-10-5-χ(|NTMap|)(|a|) :
    ?cf-Cone(|ObjMap|)(|a|) ↦iso cat-Set β ?L-10-5-N(|ObjMap|)(|a|)
    if a ∈∘ 𝔄(|Obj|) for a
    using assms(2,3,4,5) that
    by
      (
        cs-concl
          cs-simp: L-10-5-χ-NTMap-app
          cs-intro: cat-cs-intros L-10-5-χ'-arrow-is-iso-arr
      )
  from cat-Set-is-iso-arrD[OF this] show
    ?L-10-5-χ(|NTMap|)(|a|) : ?cf-Cone(|ObjMap|)(|a|) ↦cat-Set β ?L-10-5-N(|ObjMap|)(|a|)
    if a ∈∘ 𝔄(|Obj|) for a
    using that by auto

  have [cat-cs-simps]:
    ?L-10-5-χ'-arrow b ∘A cat-Set β
      cf-hom ?cℜ-𝔄 [ntcf-arrow (?ntcf-cℜ-𝔄 f), ntcf-arrow (ntcf-id ?T-cℜ)]∘ =
      cf-hom (?FUNCT 𝔅)
        [
          ntcf-arrow (ntcf-id (?H-ℭ c ∘CF ℜ)),
          ntcf-arrow (HomA.Cα𝔄(f,−) ∘NTCF-CF T)
        ]∘ ∘A cat-Set β ?L-10-5-χ'-arrow a
  (
    is
      ?L-10-5-χ'-arrow b ∘A cat-Set β ?cf-hom-lhs =
        ?cf-hom-rhs ∘A cat-Set β ?L-10-5-χ'-arrow a
  )
    if f : b ↦𝔄 a for a b f
  proof–
    let ?H-f = ‹HomA.Cα𝔄(f,−)›
    from that assms cℜ-𝔄.category-axioms cℜ-𝔄.category-axioms have lhs:
      ?L-10-5-χ'-arrow b ∘A cat-Set β ?cf-hom-lhs :
        Hom ?cℜ-𝔄 (cf-map (?cf-cℜ-𝔄 a)) (cf-map ?T-cℜ) ↦cat-Set β
        ?L-10-5-N(|ObjMap|)(|b|)
      by
        (
```

```
      cs-concl
        cs-simp:
          cat-Kan-cs-simps
          cat-cs-simps
          cat-FUNCT-cs-simps
          cat-FUNCT-components(1)
          cat-op-simps
        cs-intro:
          cat-Kan-cs-intros
          cat-FUNCT-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          cat-op-intros
    )
then have dom-lhs:
  𝒟∘ ((?L-10-5-χ'-arrow b ∘_A cat-Set β ?cf-hom-lhs)(|ArrVal|)) =
    Hom ?cℜ-𝔄 (cf-map (?cf-cℜ-𝔄 a)) (cf-map ?𝔗-cℜ)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
from that assms cℜ-𝔄.category-axioms cℜ-𝔄.category-axioms have rhs:
  ?cf-hom-rhs ∘_A cat-Set β ?L-10-5-χ'-arrow a :
    Hom ?cℜ-𝔄 (cf-map (?cf-cℜ-𝔄 a)) (cf-map ?𝔗-cℜ) ↦_cat-Set β
    ?L-10-5-N(|ObjMap|)(|b|)
  by
    (
      cs-concl
        cs-simp:
          cat-Kan-cs-simps
          cat-cs-simps
          cat-FUNCT-components(1)
          cat-op-simps
        cs-intro:
          cat-Kan-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          cat-FUNCT-cs-intros
          cat-op-intros
    )
then have dom-rhs:
  𝒟∘ ((?cf-hom-rhs ∘_A cat-Set β ?L-10-5-χ'-arrow a)(|ArrVal|)) =
    Hom ?cℜ-𝔄 (cf-map (?cf-cℜ-𝔄 a)) (cf-map ?𝔗-cℜ)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)

show ?thesis
proof(rule arr-Set-eqI)
  from lhs show arr-Set-lhs:
    arr-Set β (?L-10-5-χ'-arrow b ∘_A cat-Set β ?cf-hom-lhs)
    by (auto dest: cat-Set-is-arrD(1))
  from rhs show arr-Set-rhs:
    arr-Set β (?cf-hom-rhs ∘_A cat-Set β ?L-10-5-χ'-arrow a)
    by (auto dest: cat-Set-is-arrD(1))
  show
    (?L-10-5-χ'-arrow b ∘_A cat-Set β ?cf-hom-lhs)(|ArrVal|) =
      (?cf-hom-rhs ∘_A cat-Set β ?L-10-5-χ'-arrow a)(|ArrVal|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
    fix F assume prems: F : cf-map (?cf-cℜ-𝔄 a) ↦_?cℜ-𝔄 cf-map ?𝔗-cℜ
    let ?F = ‹ntcf-of-ntcf-arrow (c ↓_CF ℜ) 𝔄 F›
    from that have [cat-cs-simps]:
```

```
  cf-of-cf-map (c ↓CF ℜ) 𝔄 (cf-map (?cf-cℜ-𝔄 a)) = ?cf-cℜ-𝔄 a
  cf-of-cf-map (c ↓CF ℜ) 𝔄 (cf-map (?𝔗-cℜ)) = ?𝔗-cℜ
  by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)
note F = cat-FUNCT-is-arrD[OF prems, unfolded cat-cs-simps]
from that F(1) have F-const-is-cat-cone:
  ?F ⁃NTCF ?ntcf-cℜ-𝔄 f : b <CF.cone ?𝔗-cℜ : c ↓CF ℜ ↦↦Cα 𝔄
  by
    (
      cs-concl
        cs-simp: cat-cs-simps cs-intro: is-cat-coneI cat-cs-intros
    )
have [cat-cs-simps]:
  ?L-10-5-υ (ntcf-arrow (?F ⁃NTCF ?ntcf-cℜ-𝔄 f)) b =
    ?H-f ∘NTCF-CF 𝔗 ⁃NTCF ?L-10-5-υ (ntcf-arrow ?F) a
proof(rule ntcf-eqI)
  from assms that F(1) show
    ?L-10-5-υ (ntcf-arrow (?F ⁃NTCF ?ntcf-cℜ-𝔄 f)) b :
      ?H-ℭ c ∘CF ℜ ↦CF ?H-𝔄 b ∘CF 𝔗 : 𝔅 ↦↦Cα cat-Set α
    by
      (
        cs-concl cs-intro:
          cat-Kan-cs-intros cat-cs-intros is-cat-coneI
      )
  then have dom-υ:
    𝒟∘ (?L-10-5-υ (ntcf-arrow (?F ⁃NTCF ?ntcf-cℜ-𝔄 f)) b(|NTMap|)) =
      𝔅(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms that F(1) show
    ?H-f ∘NTCF-CF 𝔗 ⁃NTCF ?L-10-5-υ (ntcf-arrow ?F) a :
      ?H-ℭ c ∘CF ℜ ↦CF ?H-𝔄 b ∘CF 𝔗 : 𝔅 ↦↦Cα cat-Set α
    by
      (
        cs-concl cs-intro:
          cat-Kan-cs-intros cat-cs-intros is-cat-coneI
      )
  then have dom-f𝔗υ:
    𝒟∘ ((?H-f ∘NTCF-CF 𝔗 ⁃NTCF ?L-10-5-υ (ntcf-arrow ?F) a)(|NTMap|)) =
      𝔅(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps)
  show
    ?L-10-5-υ (ntcf-arrow (?F ⁃NTCF ?ntcf-cℜ-𝔄 f)) b(|NTMap|) =
      (?H-f ∘NTCF-CF 𝔗 ⁃NTCF ?L-10-5-υ (ntcf-arrow ?F) a)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-υ dom-f𝔗υ)
    fix b′ assume prems′: b′ ∈∘ 𝔅(|Obj|)
    let ?Y = ‹Yoneda-component (?H-𝔄 b) a f (𝔗(|ObjMap|)(|b′|))›
    let ?ℜb′ = ‹ℜ(|ObjMap|)(|b′|)›
    let ?𝔗b′ = ‹𝔗(|ObjMap|)(|b′|)›
    have [cat-cs-simps]:
      ?L-10-5-υ-arrow (ntcf-arrow (?F ⁃NTCF ?ntcf-cℜ-𝔄 f)) b b′ =
        ?Y ∘A cat-Set α ?L-10-5-υ-arrow (ntcf-arrow ?F) a b′
      (is ‹?υ-Ffbb′ = ?Yυ›)
    proof–
      from assms prems′ F-const-is-cat-cone have υ-Ffbb′:
        ?υ-Ffbb′ : Hom ℭ c ?ℜb′ ↦cat-Set α Hom 𝔄 b ?𝔗b′
        by
          (
            cs-concl cs-shallow
              cs-intro: cat-cs-intros L-10-5-υ-arrow-is-arr
```

```
    )
  then have dom-υ-Ffbb': 𝒟∘ (?υ-Ffbb'(|ArrVal|)) = Hom ℭ c (?ℜb')
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms that 𝔗.HomCod.category-axioms prems' F(1) have Yυ:
    ?Yυ : Hom ℭ c ?ℜb' ↦_cat-Set α Hom 𝔄 b ?𝔗b'
    by
      (
        cs-concl
          cs-simp: cat-Kan-cs-simps cat-cs-simps cat-op-simps
          cs-intro: is-cat-coneI cat-Kan-cs-intros cat-cs-intros
      )
  then have dom-Yυ: 𝒟∘ (?Yυ(|ArrVal|)) = Hom ℭ c (?ℜb')
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
  proof(rule arr-Set-eqI)
    from υ-Ffbb' show arr-Set-υ-Ffbb': arr-Set α ?υ-Ffbb'
      by (auto dest: cat-Set-is-arrD(1))
    from Yυ show arr-Set-Yυ: arr-Set α ?Yυ
      by (auto dest: cat-Set-is-arrD(1))
    show ?υ-Ffbb'(|ArrVal|) = ?Yυ(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-υ-Ffbb' dom-Yυ in-Hom-iff)
      fix g assume g : c ↦_ℭ ?ℜb'
      with
        assms(2–)
        ℜ.is-functor-axioms
        𝔗.is-functor-axioms
        𝔗.HomCod.category-axioms
        ℜ.HomCod.category-axioms
        that prems' F(1)
      show ?υ-Ffbb'(|ArrVal|)(|g|) = ?Yυ(|ArrVal|)(|g|)
        by
          (
            cs-concl
              cs-simp:
                cat-Kan-cs-simps
                cat-cs-simps
                L-10-5-υ-arrow-ArrVal-app
                cat-comma-cs-simps
                cat-op-simps
              cs-intro:
                cat-Kan-cs-intros
                is-cat-coneI
                cat-cs-intros
                cat-comma-cs-intros
                cat-op-intros
              cs-simp: cat-FUNCT-cs-simps
          )
    qed (use arr-Set-υ-Ffbb' arr-Set-Yυ in auto)
  qed
    (
      use υ-Ffbb' Yυ in
        ‹cs-concl cs-shallow cs-simp: cat-cs-simps›
    )+
qed

from assms prems' that F(1) show
  ?L-10-5-υ (ntcf-arrow (?F ·_NTCF ?ntcf-cℜ-𝔄 f)) b(|NTMap|)(|b'|) =
    (?H-f ∘_NTCF-CF 𝔗 ·_NTCF ?L-10-5-υ (ntcf-arrow ?F) a)(|NTMap|)(|b'|)
```

```
          by
            (
              cs-concl
                cs-simp: cat-Kan-cs-simps cat-cs-simps
                cs-intro: is-cat-coneI cat-Kan-cs-intros cat-cs-intros
            )

        qed (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)+

      qed simp-all

      from that F(1) interpret F: is-cat-cone α a ‹c ↓_{C F} ℜ› 𝔄 ‹?𝔗-cℜ› ?F
        by (cs-concl cs-shallow cs-intro: is-cat-coneI cat-cs-intros)
      from
        assms(2–) prems F(1) that
        𝔗.HomCod.cat-ntcf-Hom-snd-is-ntcf[OF that]
        cℜ-𝔄.category-axioms
      show
        (?L-10-5-χ'-arrow b ∘_{A cat-Set β} ?cf-hom-lhs)(|ArrVal|)(|F|) =
          (?cf-hom-rhs ∘_{A cat-Set β} ?L-10-5-χ'-arrow a)(|ArrVal|)(|F|)
        by (subst (1 2) F(2))
          (
            cs-concl
              cs-simp:
                cat-cs-simps
                cat-Kan-cs-simps
                cat-FUNCT-cs-simps
                cat-FUNCT-components(1)
                cat-op-simps
              cs-intro:
                is-cat-coneI
                cat-Kan-cs-intros
                cat-cs-intros
                cat-prod-cs-intros
                cat-FUNCT-cs-intros
                cat-op-intros
          )
    qed (use arr-Set-lhs arr-Set-rhs in auto)

  qed (use lhs rhs in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed

show
  ?L-10-5-χ(|NTMap|)(|b|) ∘_{A cat-Set β} ?cf-Cone(|ArrMap|)(|f|) =
    ?L-10-5-N(|ArrMap|)(|f|) ∘_{A cat-Set β} ?L-10-5-χ(|NTMap|)(|a|)
  if f : b ↦_𝔄 a for a b f
  using that assms
  by
    (
      cs-concl
        cs-simp:
          cat-cs-simps
          cat-Kan-cs-simps
          cat-FUNCT-components(1)
          cat-FUNCT-cs-simps
          cat-op-simps
        cs-intro:
```

cat-Kan-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
)

qed
(
cs-concl
cs-simp: *cat-Kan-cs-simps* **cs-intro**: *cat-cs-intros cat-Kan-cs-intros*
)+

qed

15.9 The existence of a canonical limit or a canonical colimit for the pointwise Kan extensions

lemma (**in** *is-cat-pw-rKe*) *cat-pw-rKe-ex-cat-limit*:
— Based on the elements of Chapter X-5 in [9].
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
obtains *UA*
where $UA : \mathfrak{G}(\!|ObjMap|\!)(\!|c|\!) <_{CF.lim} \mathfrak{T} \circ_{CF} c\ {}_{O}\sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–

define β **where** $\beta = \alpha + \omega$
have β: $\mathcal{Z}\ \beta$ **and** $\alpha\beta$: $\alpha \in_{\circ} \beta$
by (*simp-all add*: β-*def AG.*$\mathcal{Z}$-*Limit-*$\alpha\omega$ *AG.*$\mathcal{Z}$-ω-$\alpha\omega$ $\mathcal{Z}$-*def AG.*$\mathcal{Z}$-α-$\alpha\omega$)
then interpret β: $\mathcal{Z}\ \beta$ **by** *simp*

let *?FUNCT* = ‹$\lambda\mathfrak{A}$. *cat-FUNCT* α $\mathfrak{A}$ (*cat-Set* α)›
let *?H-A* = ‹λf. $Hom_{A.C\alpha}\mathfrak{A}(f,-)$›
let *?H-A*$\mathfrak{G}$ = ‹λf. *?H-A* f $\circ_{NTCF-CF}$ $\mathfrak{G}$›
let *?H-*$\mathfrak{A}$ = ‹λa. $Hom_{O.C\alpha}\mathfrak{A}(a,-)$›
let *?H-*$\mathfrak{A}\mathfrak{T}$ = ‹λa. *?H-*$\mathfrak{A}$ a $\circ_{CF}$ $\mathfrak{T}$›
let *?H-*$\mathfrak{A}\mathfrak{G}$ = ‹λa. *?H-*$\mathfrak{A}$ a $\circ_{CF}$ $\mathfrak{G}$›
let *?H-*$\mathfrak{C}$ = ‹λc. $Hom_{O.C\alpha}\mathfrak{C}(c,-)$›
let *?H-*$\mathfrak{C}\mathfrak{K}$ = ‹λc. *?H-*$\mathfrak{C}$ c $\circ_{CF}$ $\mathfrak{K}$›
let *?H-*$\mathfrak{A}\varepsilon$ = ‹λb. *?H-*$\mathfrak{A}$ b $\circ_{CF-NTCF}$ ε›
let *?SET-*$\mathfrak{K}$ = ‹*exp-cat-cf* α (*cat-Set* α) $\mathfrak{K}$›
let *?H-FUNCT* = ‹$\lambda\mathfrak{C}$ $\mathfrak{F}$. $Hom_{O.C\beta}$*?FUNCT* $\mathfrak{C}$(−,*cf-map* $\mathfrak{F}$)›
let *?ua-NTDGDom* = ‹*op-cat* (*?FUNCT* $\mathfrak{C}$)›
let *?ua-NTDom* = ‹λa. *?H-FUNCT* $\mathfrak{C}$ (*?H-*$\mathfrak{A}\mathfrak{G}$ a)›
let *?ua-NTCod* = ‹λa. *?H-FUNCT* $\mathfrak{B}$ (*?H-*$\mathfrak{A}\mathfrak{T}$ a) $\circ_{CF}$ *op-cf ?SET-*$\mathfrak{K}$›
let *?c*$\mathfrak{K}$*-*$\mathfrak{A}$ = ‹*cat-FUNCT* α ($c \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A}$›
let *?ua* =
‹
λa. *ntcf-ua-fo*
β
?SET-$\mathfrak{K}$
(*cf-map* (*?H-*$\mathfrak{A}\mathfrak{T}$ a))
(*cf-map* (*?H-*$\mathfrak{A}\mathfrak{G}$ a))
(*ntcf-arrow* (*?H-*$\mathfrak{A}\varepsilon$ a))
›
let *?cf-nt* = ‹*cf-nt* α β (*cf-id* $\mathfrak{C}$)›
let *?cf-eval* = ‹*cf-eval* α β $\mathfrak{C}$›
let *?*$\mathfrak{T}$*-c*$\mathfrak{K}$ = ‹$\mathfrak{T} \circ_{CF} c\ {}_{O}\sqcap_{CF} \mathfrak{K}$›

let *?cf-cℜ-𝔄* = ‹*cf-const* (*c* ↓$_{CF}$ ℜ) 𝔄›
let *?𝔊c* = ‹𝔊(|*ObjMap*|)(|*c*|)›
let *?Δ* = ‹Δ$_{CF}$ *α* (*c* ↓$_{CF}$ ℜ) 𝔄›
let *?ntcf-ua-fo* =
  ‹
    λa. ntcf-ua-fo
      β
      ?SET-ℜ
      (*cf-map* (*?H-𝔄𝔗 a*))
      (*cf-map* (*?H-𝔄𝔊 a*))
      (*ntcf-arrow* (*?H-𝔄ε a*))
  ›
let *?umap-fo* =
  ‹
    λb. umap-fo
      ?SET-ℜ
      (*cf-map* (*?H-𝔄𝔗 b*))
      (*cf-map* (*?H-𝔄𝔊 b*))
      (*ntcf-arrow* (*?H-𝔄ε b*))
      (*cf-map* (*?H-𝔈 c*))
  ›

interpret ℜ: *is-functor α 𝔅 𝔈 ℜ* **by** (*rule assms*(*1*))
interpret 𝔗: *is-functor α 𝔅 𝔄 𝔗* **by** (*rule assms*(*2*))

from *AG.vempty-is-zet assms*(*3*) **interpret** *c*ℜ: *category α ‹c* ↓$_{CF}$ ℜ›
  by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
from *αβ assms*(*3*) **interpret** *c*ℜ-𝔄: *category β ?c*ℜ-𝔄
  by
    (
      cs-concl **cs-intro**:
        cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)
from *αβ assms*(*3*) **interpret** Δ: *is-functor β 𝔄 ?c*ℜ-𝔄 *?Δ*
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)+
from *AG.vempty-is-zet assms*(*3*) **interpret** Π*c*:
  is-functor α ‹c ↓$_{CF}$ ℜ› 𝔅 ‹*c* $_O\prod_{CF}$ ℜ›
  by
    (
      cs-concl **cs-shallow**
        cs-simp: *cat-comma-cs-simps*
        cs-intro: *cat-cs-intros cat-comma-cs-intros*
)

interpret *β*Π*c*: *is-tiny-functor β ‹c* ↓$_{CF}$ ℜ› 𝔅 ‹*c* $_O\prod_{CF}$ ℜ›
  by (*rule* Π*c.cf-is-tiny-functor-if-ge-Limit*[*OF β αβ*])

interpret *E*: *is-functor β ‹?FUNCT* (𝔈 ×$_C$ 𝔈)› ‹*cat-Set β*› *?cf-eval*
  by (*rule AG.HomCod.cat-cf-eval-is-functor*[*OF β αβ*])

from *αβ* **interpret** *FUNCT-𝔄*: *tiny-category β ‹?FUNCT 𝔄*›
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *αβ* **interpret** *FUNCT-𝔅*: *tiny-category β ‹?FUNCT 𝔅*›
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *αβ* **interpret** *FUNCT-𝔈*: *tiny-category β ‹?FUNCT 𝔈*›
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)

interpret *β𝔄*: *tiny-category β 𝔄*

by (*rule category.cat-tiny-category-if-ge-Limit*)
(*use* $\alpha\beta$ **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{B}$: *tiny-category* β $\mathfrak{B}$
by (*rule category.cat-tiny-category-if-ge-Limit*)
(*use* $\alpha\beta$ **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{C}$: *tiny-category* β $\mathfrak{C}$
by (*rule category.cat-tiny-category-if-ge-Limit*)
(*use* $\alpha\beta$ **in** ‹*cs-concl cs-intro*: *cat-cs-intros*›)+

interpret $\beta\mathfrak{K}$: *is-tiny-functor* β $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$
by (*rule is-functor.cf-is-tiny-functor-if-ge-Limit*)
(*use* $\alpha\beta$ **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{G}$: *is-tiny-functor* β $\mathfrak{C}$ $\mathfrak{A}$ $\mathfrak{G}$
by (*rule is-functor.cf-is-tiny-functor-if-ge-Limit*)
(*use* $\alpha\beta$ **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{T}$: *is-tiny-functor* β $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$
by (*rule is-functor.cf-is-tiny-functor-if-ge-Limit*)
(*use* $\alpha\beta$ **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+

interpret *cat-Set-*$\alpha\beta$: *subcategory* β ‹*cat-Set* α› ‹*cat-Set* β›
by (*rule AG.subcategory-cat-Set-cat-Set*[*OF* β $\alpha\beta$])

from *assms*(*3*) $\alpha\beta$ **interpret** *Hom-c*: *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› ‹*?H-*$\mathfrak{C}$ *c*›
by (*cs-concl* **cs-intro**: *cat-cs-intros*)

define E' :: V **where** E' =
[
($\lambda a \in_\circ \mathfrak{A}(\!|Obj|\!)$. *?cf-eval*(*ObjMap*)(*cf-map* (*?H-*$\mathfrak{A}\mathfrak{G}$ *a*), *c*)$_\bullet$),
($\lambda f \in_\circ \mathfrak{A}(\!|Arr|\!)$. *?cf-eval*(*ArrMap*)(*ntcf-arrow* (*?H-A*$\mathfrak{G}$ *f*), $\mathfrak{C}$(*CId*)(*c*)$_\bullet$),
op-cat $\mathfrak{A}$,
cat-Set β
]$_\circ$

have E'*-components*:
$E'(\!|ObjMap|\!)$ = ($\lambda a \in_\circ \mathfrak{A}(\!|Obj|\!)$. *?cf-eval*(*ObjMap*)(*cf-map* (*?H-*$\mathfrak{A}\mathfrak{G}$ *a*), *c*)$_\bullet$)
$E'(\!|ArrMap|\!)$ =
($\lambda f \in_\circ \mathfrak{A}(\!|Arr|\!)$. *?cf-eval*(*ArrMap*)(*ntcf-arrow* (*?H-A*$\mathfrak{G}$ *f*), $\mathfrak{C}$(*CId*)(*c*)$_\bullet$)
$E'(\!|HomDom|\!)$ = *op-cat* $\mathfrak{A}$
$E'(\!|HomCod|\!)$ = *cat-Set* β
unfolding E'*-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

note [*cat-cs-simps*] = E'*-components*(*3*,*4*)

have E'*-ObjMap-app*[*cat-cs-simps*]:
$E'(\!|ObjMap|\!)(\!|a|\!)$ = *?cf-eval*(*ObjMap*)(*cf-map* (*?H-*$\mathfrak{A}\mathfrak{G}$ *a*), *c*)$_\bullet$
if $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **for** *a*
using *that* **unfolding** E'*-components* **by** *simp*
have E'*-ArrMap-app*[*cat-cs-simps*]:
$E'(\!|ArrMap|\!)(\!|f|\!)$ = *?cf-eval*(*ArrMap*)(*ntcf-arrow* (*?H-A*$\mathfrak{G}$ *f*), $\mathfrak{C}$(*CId*)(*c*)$_\bullet$
if $f \in_\circ \mathfrak{A}(\!|Arr|\!)$ **for** *f*
using *that* **unfolding** E'*-components* **by** *simp*

have E': E' : *op-cat* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\beta}$ *cat-Set* β
proof(*intro is-functorI'*)

show *vfsequence* E' **unfolding** E'*-def* **by** *auto*

show *vcard* $E' = \mathcal{4}_{\mathbb{N}}$ **unfolding** *E′-def* **by** (*simp add*: *nat-omega-simps*)
show *vsv* ($E'(\!|ObjMap|\!)$) **unfolding** *E′-components* **by** *simp*
show *vsv* ($E'(\!|ArrMap|\!)$) **unfolding** *E′-components* **by** *simp*
show $\mathcal{D}_{\circ}$ ($E'(\!|ObjMap|\!)$) = *op-cat* $\mathfrak{A}(\!|Obj|\!)$
  unfolding *E′-components* **by** (*simp add*: *cat-op-simps*)
show $\mathcal{R}_{\circ}$ ($E'(\!|ObjMap|\!)$) $\subseteq_{\circ}$ *cat-Set* $\beta(\!|Obj|\!)$
  unfolding *E′-components*
proof(*rule vrange-VLambda-vsubset*)
  fix *a* **assume** *prems*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
  then have *?H-𝔄𝔊* $a : \mathfrak{C} \mapsto\!\!\mapsto_{C\alpha}$ *cat-Set* α
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  with *assms*(*3*) *prems* **show**
    ?cf-eval($\!|ObjMap|\!$)($\!|$*cf-map* (*?H-𝔄𝔊* *a*), *c*$|\!|_{\bullet}$ $\in_{\circ}$ *cat-Set* $\beta(\!|Obj|\!)$
    by
      (
        cs-concl
          cs-simp: *cat-cs-simps cat-Set-components*(*1*)
          cs-intro: *cat-cs-intros cat-op-intros Ran.HomCod.cat-Hom-in-Vset*
)
qed
show $\mathcal{D}_{\circ}$ ($E'(\!|ArrMap|\!)$) = *op-cat* $\mathfrak{A}(\!|Arr|\!)$
  unfolding *E′-components* **by** (*simp add*: *cat-op-simps*)
show $E'(\!|ArrMap|\!)(\!|f|\!) : E'(\!|ObjMap|\!)(\!|a|\!) \mapsto_{cat\text{-}Set\ \beta} E'(\!|ObjMap|\!)(\!|b|\!)$
  if $f : a \mapsto_{op\text{-}cat\ \mathfrak{A}} b$ **for** *a b f*
proof–
  from *that*[*unfolded cat-op-simps*] *assms*(*3*) **show** *?thesis*
    by (*intro cat-Set-αβ.subcat-is-arrD*)
      (
        cs-concl
          cs-simp:
            category.cf-eval-ObjMap-app
            category.cf-eval-ArrMap-app
            E′-ObjMap-app
            E′-ArrMap-app
          cs-intro: *cat-cs-intros*
)
qed
then have [*cat-cs-intros*]: $E'(\!|ArrMap|\!)(\!|f|\!) : A \mapsto_{cat\text{-}Set\ \beta} B$
  if $A = E'(\!|ObjMap|\!)(\!|a|\!)$ **and** $B = E'(\!|ObjMap|\!)(\!|b|\!)$ **and** $f : b \mapsto_{\mathfrak{A}} a$
  for *a b f A B*
  using *that* **by** (*simp add*: *cat-op-simps*)
show
  $E'(\!|ArrMap|\!)(\!|g \circ_{A\ op\text{-}cat\ \mathfrak{A}} f|\!) = E'(\!|ArrMap|\!)(\!|g|\!) \circ_{A\ cat\text{-}Set\ \beta} E'(\!|ArrMap|\!)(\!|f|\!)$
  if $g : b \mapsto_{op\text{-}cat\ \mathfrak{A}} c$ **and** $f : a \mapsto_{op\text{-}cat\ \mathfrak{A}} b$ **for** *b c g a f*
proof–
  note g = *that*(*1*)[*unfolded cat-op-simps*]
    and f = *that*(*2*)[*unfolded cat-op-simps*]
  from *g f assms*(*3*) $\alpha\beta$ **show** *?thesis*
    by
      (
        cs-concl
          cs-intro:
            cat-cs-intros
            cat-prod-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
          cs-simp:
            cat-cs-simps

```
            cat-FUNCT-cs-simps
            cat-prod-cs-simps
            cat-op-simps
            E.cf-ArrMap-Comp[symmetric]
        )+
  qed

  show E'(|ArrMap|)(|op-cat 𝔄(|CId|)(|a|)|) = cat-Set β(|CId|)(|E'(|ObjMap|)(|a|)|)
    if a ∈∘ op-cat 𝔄(|Obj|) for a
  proof(cs-concl-step cat-Set-αβ.subcat-CId[symmetric])
    from that[unfolded cat-op-simps] assms(3) show
      E'(|ObjMap|)(|a|) ∈∘ cat-Set α(|Obj|)
      by
        (
          cs-concl
            cs-simp: cat-Set-components(1) cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros
        )
    from that[unfolded cat-op-simps] assms(3) show
      E'(|ArrMap|)(|op-cat 𝔄(|CId|)(|a|)|) = cat-Set α(|CId|)(|E'(|ObjMap|)(|a|)|)
      by
        (
          cs-concl
            cs-intro: cat-cs-intros
            cs-simp:
              cat-Set-components(1)
              cat-cs-simps
              cat-op-simps
              ntcf-id-cf-comp[symmetric]
        )
  qed
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
then interpret E': is-functor β ‹op-cat 𝔄› ‹cat-Set β› E' by simp




define N' :: V where N' =
  [
    (λa∈∘𝔄(|Obj|). ?cf-nt(|ObjMap|)(|cf-map (?H-𝔄𝔊 a), c|)•),
    (λf∈∘𝔄(|Arr|). ?cf-nt(|ArrMap|)(|ntcf-arrow (?H-A𝔊 f), ℭ(|CId|)(|c|)|)•),
    op-cat 𝔄,
    cat-Set β
  ]∘

have N'-components:
  N'(|ObjMap|) = (λa∈∘𝔄(|Obj|). ?cf-nt(|ObjMap|)(|cf-map (?H-𝔄𝔊 a), c|)•)
  N'(|ArrMap|) =
    (λf∈∘𝔄(|Arr|). ?cf-nt(|ArrMap|)(|ntcf-arrow (?H-A𝔊 f), ℭ(|CId|)(|c|)|)•)
  N'(|HomDom|) = op-cat 𝔄
  N'(|HomCod|) = cat-Set β
  unfolding N'-def dghm-field-simps by (simp-all add: nat-omega-simps)

note [cat-cs-simps] = N'-components(3,4)

have N'-ObjMap-app[cat-cs-simps]:
  N'(|ObjMap|)(|a|) = ?cf-nt(|ObjMap|)(|cf-map (?H-𝔄𝔊 a), c|)•
  if a ∈∘ 𝔄(|Obj|) for a
```

```
  using that unfolding N'-components by simp
have N'-ArrMap-app[cat-cs-simps]:
  N'(|ArrMap|)(|f|) = ?cf-nt(|ArrMap|)(|ntcf-arrow (?H-AG f), C(|CId|)(|c|)|)•
  if f ∈∘ A(|Arr|) for f
  using that unfolding N'-components by simp

from αβ interpret cf-nt-C: is-functor β ‹?FUNCT C ×C C› ‹cat-Set β› ‹?cf-nt›
  by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

have N': N' : op-cat A ↦↦C β cat-Set β
proof(intro is-functorI')
  show vfsequence N' unfolding N'-def by simp
  show vcard N' = 4ℕ unfolding N'-def by (simp add: nat-omega-simps)
  show vsv (N'(|ObjMap|)) unfolding N'-components by simp
  show vsv (N'(|ArrMap|)) unfolding N'-components by simp
  show D∘ (N'(|ObjMap|)) = op-cat A(|Obj|)
    unfolding N'-components by (simp add: cat-op-simps)
  show R∘ (N'(|ObjMap|)) ⊆∘ cat-Set β(|Obj|)
    unfolding N'-components
  proof(rule vrange-VLambda-vsubset)
    fix a assume prems: a ∈∘ A(|Obj|)
    with assms(3) αβ show
      ?cf-nt(|ObjMap|)(|cf-map (?H-AG a), c|)• ∈∘ cat-Set β(|Obj|)
      by
        (
          cs-concl
            cs-simp: cat-Set-components(1) cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros FUNCT-C.cat-Hom-in-Vset cat-FUNCT-cs-intros
        )
  qed
  show D∘ (N'(|ArrMap|)) = op-cat A(|Arr|)
    unfolding N'-components by (simp add: cat-op-simps)
  show N'(|ArrMap|)(|f|) : N'(|ObjMap|)(|a|) ↦cat-Set β N'(|ObjMap|)(|b|)
    if f : a ↦op-cat A b for a b f
    using that[unfolded cat-op-simps] assms(3)
    by
      (
        cs-concl
          cs-simp: N'-ObjMap-app N'-ArrMap-app
          cs-intro: cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros
      )
  show
    N'(|ArrMap|)(|g ∘A op-cat A f|) = N'(|ArrMap|)(|g|) ∘A cat-Set β N'(|ArrMap|)(|f|)
    if g : b ↦op-cat A c and f : a ↦op-cat A b for b c g a f
  proof–
    from that assms(3) αβ show ?thesis
      unfolding cat-op-simps
      by
        (
          cs-concl
            cs-intro:
              cat-cs-intros
              cat-prod-cs-intros
              cat-FUNCT-cs-intros
              cat-op-intros
            cs-simp:
              cat-cs-simps
              cat-FUNCT-cs-simps
```

```
        cat-prod-cs-simps
        cat-op-simps
        cf-nt-ℭ.cf-ArrMap-Comp[symmetric]
    )
  qed
  show N'(|ArrMap|)(|op-cat 𝔄(|CId|)(|a|)|) = cat-Set β(|CId|)(|N'(|ObjMap|)(|a|)|)
    if a ∈∘ op-cat 𝔄(|Obj|) for a
  proof–
    note [cat-cs-simps] =
      ntcf-id-cf-comp[symmetric]
      ntcf-arrow-id-ntcf-id[symmetric]
      cat-FUNCT-CId-app[symmetric]
    from that[unfolded cat-op-simps] assms(3) αβ show ?thesis
      by
        (
          cs-concl
            cs-intro:
              cat-cs-intros
              cat-FUNCT-cs-intros
              cat-prod-cs-intros
              cat-op-intros
            cs-simp: cat-FUNCT-cs-simps cat-cs-simps cat-op-simps
        )+
  qed
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
then interpret N': is-functor β ‹op-cat 𝔄› ‹cat-Set β› N' by simp




define Y' :: V where Y' =
  [
    (λa∈∘𝔄(|Obj|). ntcf-Yoneda α β ℭ(|NTMap|)(|cf-map (?H-𝔄𝔊 a), c|)•),
    N',
    E',
    op-cat 𝔄,
    cat-Set β
  ]∘

have Y'-components:
  Y'(|NTMap|) = (λa∈∘𝔄(|Obj|). ntcf-Yoneda α β ℭ(|NTMap|)(|cf-map (?H-𝔄𝔊 a), c|)•)
  Y'(|NTDom|) = N'
  Y'(|NTCod|) = E'
  Y'(|NTDGDom|) = op-cat 𝔄
  Y'(|NTDGCod|) = cat-Set β
  unfolding Y'-def nt-field-simps by (simp-all add: nat-omega-simps)

note [cat-cs-simps] = Y'-components(2–5)

have Y'-NTMap-app[cat-cs-simps]:
  Y'(|NTMap|)(|a|) = ntcf-Yoneda α β ℭ(|NTMap|)(|cf-map (?H-𝔄𝔊 a), c|)•
  if a ∈∘ 𝔄(|Obj|) for a
  using that unfolding Y'-components by simp

from β αβ interpret Y:
  is-iso-ntcf β ‹?FUNCT ℭ ×C ℭ› ‹cat-Set β› ?cf-nt ?cf-eval ‹ntcf-Yoneda α β ℭ›
  by (rule AG.HomCod.cat-ntcf-Yoneda-is-ntcf)
```

```
have Y': Y' : N' ↦_{CF.iso} E' : op-cat 𝔄 ↦↦_{Cβ} cat-Set β
proof(intro is-iso-ntcfI is-ntcfI')

  show vfsequence Y' unfolding Y'-def by simp
  show vcard Y' = 5_ℕ
    unfolding Y'-def by (simp add: nat-omega-simps)
  show vsv (Y'(|NTMap|)) unfolding Y'-components by auto
  show 𝒟_∘ (Y'(|NTMap|)) = op-cat 𝔄(|Obj|)
    unfolding Y'-components by (simp add: cat-op-simps)
  show Y'-NTMap-a: Y'(|NTMap|)(|a|) : N'(|ObjMap|)(|a|) ↦_{iso cat-Set β} E'(|ObjMap|)(|a|)
    if a ∈_∘ op-cat 𝔄(|Obj|) for a
    using that[unfolded cat-op-simps] assms(3) αβ
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-arrow-cs-intros
            cat-cs-intros
            cat-prod-cs-intros
            cat-FUNCT-cs-intros
      )
  then show Y'(|NTMap|)(|a|) : N'(|ObjMap|)(|a|) ↦_{cat-Set β} E'(|ObjMap|)(|a|)
    if a ∈_∘ op-cat 𝔄(|Obj|) for a
    by (intro cat-Set-is-iso-arrD[OF Y'-NTMap-a[OF that]])
  show
    Y'(|NTMap|)(|b|) ∘_{A cat-Set β} N'(|ArrMap|)(|f|) =
      E'(|ArrMap|)(|f|) ∘_{A cat-Set β} Y'(|NTMap|)(|a|)
    if f : a ↦_{op-cat 𝔄} b for a b f
  proof–
    note f = that[unfolded cat-op-simps]
    from f assms(3) show ?thesis
      by
        (
          cs-concl
            cs-simp: cat-cs-simps Y.ntcf-Comp-commute
            cs-intro: cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros
        )+
  qed
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

have E'-def: E' = Hom_{O.Cβ}𝔄(−,?𝔊c)
proof(rule cf-eqI)
  show E' : op-cat 𝔄 ↦↦_{Cβ} cat-Set β
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from assms(3) show
    Hom_{O.Cβ}𝔄(−,?𝔊c) : op-cat 𝔄 ↦↦_{Cβ} cat-Set β
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  have dom-lhs: 𝒟_∘ (E'(|ObjMap|)) = 𝔄(|Obj|) unfolding E'-components by simp
  from assms(3) have dom-rhs:
    𝒟_∘ (Hom_{O.Cβ}𝔄(−,?𝔊c)(|ObjMap|)) = 𝔄(|Obj|)
    unfolding E'-components
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
      )
```

```
show E'(|ObjMap|) = Hom_{O.CβA}(−,?Gc)(|ObjMap|)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  fix a assume a ∈∘ A(|Obj|)
  with assms(3) show E'(|ObjMap|)(|a|) = Hom_{O.CβA}(−,?Gc)(|ObjMap|)(|a|)
    by
      (
        cs-concl
          cs-simp: cat-op-simps cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
qed (auto simp: E'-components cat-cs-intros assms(3))

have dom-lhs: D∘ (E'(|ArrMap|)) = A(|Arr|) unfolding E'-components by simp
from assms(3) have dom-rhs:
  D∘ (Hom_{O.CβA}(−,?Gc)(|ArrMap|)) = A(|Arr|)
  unfolding E'-components
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
    )

show E'(|ArrMap|) = Hom_{O.CβA}(−,?Gc)(|ArrMap|)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)

  fix f assume prems: f ∈∘ A(|Arr|)
  then obtain a b where f: f : a ↦_A b by auto
  have [cat-cs-simps]:
    cf-eval-arrow C (ntcf-arrow (?H-AG f)) (C(|CId|)(|c|)) =
      cf-hom A [f, A(|CId|)(|?Gc|)]∘
    (is ‹?cf-eval-arrow = ?cf-hom-fGc›)
  proof–
    have cf-eval-arrow-f-CId-Gc:
      ?cf-eval-arrow :
        Hom A b ?Gc ↦_{cat-Set α} Hom A a ?Gc
    proof(rule cf-eval-arrow-is-arr')
      from f show ?H-AG f : ?H-AA b ↦_{CF} ?H-AA a : C ↦↦_{Cα} cat-Set α
        by (cs-concl cs-intro: cat-cs-intros)
    qed
      (
        use f assms(3) in
          ‹
            cs-concl
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
          ›
      )+
    from f assms(3) have dom-lhs:
      D∘ (?cf-eval-arrow(|ArrVal|)) = Hom A b ?Gc
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
        )
    from assms(3) f Ran.HomCod.category-axioms have cf-hom-fGc:
      ?cf-hom-fGc :
        Hom A b ?Gc ↦_{cat-Set α} Hom A a ?Gc
      by
        (
```

```
        cs-concl cs-shallow cs-intro:
          cat-cs-intros cat-prod-cs-intros cat-op-intros
      )
    from f assms(3) have dom-rhs:
      𝒟∘ (?cf-hom-f𝔊c(|ArrVal|)) = Hom 𝔄 b ?𝔊c
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
        )

    show ?thesis
    proof(rule arr-Set-eqI)
      from cf-eval-arrow-f-CId-𝔊c show arr-Set α ?cf-eval-arrow
        by (auto dest: cat-Set-is-arrD(1))
      from cf-hom-f𝔊c show arr-Set α ?cf-hom-f𝔊c
        by (auto dest: cat-Set-is-arrD(1))
      show ?cf-eval-arrow(|ArrVal|) = ?cf-hom-f𝔊c(|ArrVal|)
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs, unfold in-Hom-iff)
        from f assms(3) show vsv (?cf-eval-arrow(|ArrVal|))
          by (cs-concl cs-intro: cat-cs-intros)
        from f assms(3) show vsv (?cf-hom-f𝔊c(|ArrVal|))
          by
            (
              cs-concl cs-shallow
                cs-simp: cat-cs-simps cat-op-simps
                cs-intro: cat-cs-intros cat-op-intros
            )
        fix g assume g : b ↦_𝔄 ?𝔊c
        with f assms(3) show
          ?cf-eval-arrow(|ArrVal|)(|g|) = ?cf-hom-f𝔊c(|ArrVal|)(|g|)
          by
            (
              cs-concl
                cs-simp: cat-cs-simps cat-op-simps
                cs-intro: cat-cs-intros cat-op-intros
            )
        qed simp

      qed
        (
          use cf-eval-arrow-f-CId-𝔊c cf-hom-f𝔊c in
            ‹cs-concl cs-simp: cat-cs-simps›
        )+

    qed

    from f prems assms(3) show E′(|ArrMap|)(|f|) = Hom_{O.Cβ}𝔄(−,?𝔊c)(|ArrMap|)(|f|)
      by
        (
          cs-concl
            cs-simp: cat-op-simps cat-cs-simps
            cs-intro: cat-cs-intros cat-op-intros
        )

  qed (auto simp: E′-components cat-cs-intros assms(3))

qed simp-all
```

from Y' **have** *inv-Y'*: *inv-ntcf* Y' :
$Hom_{O.C\beta}\mathfrak{A}(-,?\mathfrak{G}c) \mapsto_{CF.iso} N' : op\text{-}cat\ \mathfrak{A} \mapsto\mapsto_{C\beta} cat\text{-}Set\ \beta$
unfolding *E'-def* **by** (*auto intro*: *iso-ntcf-is-iso-arr*)

interpret N'': *is-functor* β ‹*op-cat* $\mathfrak{A}$› ‹*cat-Set* β› ‹*L-10-5-N* α β $\mathfrak{T}$ $\mathfrak{K}$ c›
by (*rule L-10-5-N-is-functor*[*OF* β $\alpha\beta$ *assms*])

define ψ :: V
where ψ =
[
($\lambda a \in_\circ \mathfrak{A}(\!|Obj|\!)$. *?ntcf-ua-fo* $a(\!|NTMap|\!)(\!|cf\text{-}map\ (?H\text{-}\mathfrak{C}\ c)|\!)$),
N',
L-10-5-N α β $\mathfrak{T}$ $\mathfrak{K}$ c,
op-cat $\mathfrak{A}$,
cat-Set β
]$_\circ$

have ψ-*components*:
$\psi(\!|NTMap|\!) = (\lambda a \in_\circ \mathfrak{A}(\!|Obj|\!).\ ?ntcf\text{-}ua\text{-}fo\ a(\!|NTMap|\!)(\!|cf\text{-}map\ (?H\text{-}\mathfrak{C}\ c)|\!))$
$\psi(\!|NTDom|\!) = N'$
$\psi(\!|NTCod|\!) = L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c$
$\psi(\!|NTDGDom|\!) = op\text{-}cat\ \mathfrak{A}$
$\psi(\!|NTDGCod|\!) = cat\text{-}Set\ \beta$
unfolding ψ-*def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

note [*cat-cs-simps*] = Y'-*components*(*2–5*)

have ψ-*NTMap-app*[*cat-cs-simps*]:
$\psi(\!|NTMap|\!)(\!|a|\!) = ?ntcf\text{-}ua\text{-}fo\ a(\!|NTMap|\!)(\!|cf\text{-}map\ (?H\text{-}\mathfrak{C}\ c)|\!)$
if $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **for** a
using *that* **unfolding** ψ-*components* **by** *simp*

have ψ: $\psi : N' \mapsto_{CF.iso} L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c : op\text{-}cat\ \mathfrak{A} \mapsto\mapsto_{C\beta} cat\text{-}Set\ \beta$
proof–

show *?thesis*
proof(*intro is-iso-ntcfI is-ntcfI'*)

show *vfsequence* ψ **unfolding** ψ-*def* **by** *auto*
show *vcard* $\psi = 5_{\mathbb{N}}$ **unfolding** ψ-*def* **by** (*simp-all add*: *nat-omega-simps*)
show $N' : op\text{-}cat\ \mathfrak{A} \mapsto\mapsto_{C\beta} cat\text{-}Set\ \beta$ **by** (*rule N'*)
show $L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c : op\text{-}cat\ \mathfrak{A} \mapsto\mapsto_{C\beta} cat\text{-}Set\ \beta$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $\psi(\!|NTDom|\!) = N'$ **unfolding** ψ-*components* **by** *simp*
show $\psi(\!|NTCod|\!) = L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c$ **unfolding** ψ-*components* **by** *simp*
show $\psi(\!|NTDGDom|\!) = op\text{-}cat\ \mathfrak{A}$ **unfolding** ψ-*components* **by** *simp*
show $\psi(\!|NTDGCod|\!) = cat\text{-}Set\ \beta$ **unfolding** ψ-*components* **by** *simp*
show *vsv* ($\psi(\!|NTMap|\!)$) **unfolding** ψ-*components* **by** *simp*
show $\mathcal{D}_\circ$ ($\psi(\!|NTMap|\!)$) = $op\text{-}cat\ \mathfrak{A}(\!|Obj|\!)$
unfolding ψ-*components* **by** (*simp add*: *cat-op-simps*)

show ψ-*NTMap-is-iso-arr*[*unfolded cat-op-simps*]:
$\psi(\!|NTMap|\!)(\!|a|\!) : N'(\!|ObjMap|\!)(\!|a|\!) \mapsto_{iso\,cat\text{-}Set\ \beta} L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{K}\ c(\!|ObjMap|\!)(\!|a|\!)$
if $a \in_\circ op\text{-}cat\ \mathfrak{A}(\!|Obj|\!)$ **for** a

```
proof–
  note a = that[unfolded cat-op-simps]
  interpret ε:
    is-cat-rKe-preserves α 𝔅 ℭ 𝔄 ‹cat-Set α› 𝔎 𝔗 𝔊 ‹?H-𝔄 a› ε
    by (rule cat-pw-rKe-preserved[OF a])
  interpret aε:
    is-cat-rKe α 𝔅 ℭ ‹cat-Set α› 𝔎 ‹?H-𝔄𝔗 a› ‹?H-𝔄𝔊 a› ‹?H-𝔄ε a›
    by (rule ε.cat-rKe-preserves)
  interpret is-iso-ntcf
    β
    ‹op-cat (?FUNCT ℭ)›
    ‹cat-Set β›
    ‹?H-FUNCT ℭ (?H-𝔄𝔊 a)›
    ‹?H-FUNCT 𝔅 (?H-𝔄𝔗 a) ∘_CF op-cf ?SET-𝔎›
    ‹?ntcf-ua-fo a›
    by (rule aε.cat-rKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit[OF β αβ])
  have cf-map (?H-ℭ c) ∈₀ ?FUNCT ℭ(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  from
    iso-ntcf-is-iso-arr[unfolded cat-op-simps, OF this]
    a assms αβ
  show ?thesis
    by
      (
        cs-prems
          cs-simp:
            cat-cs-simps cat-Kan-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-Kan-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
      )
qed
show ψ-NTMap-is-arr[unfolded cat-op-simps]:
  ψ(|NTMap|)(|a|) : N'(|ObjMap|)(|a|) ↦_cat-Set β L-10-5-N α β 𝔗 𝔎 c(|ObjMap|)(|a|)
  if a ∈₀ op-cat 𝔄(|Obj|) for a
  by
    (
      rule cat-Set-is-iso-arrD[
        OF ψ-NTMap-is-iso-arr[OF that[unfolded cat-op-simps]]
        ]
    )

show
  ψ(|NTMap|)(|b|) ∘_A cat-Set β N'(|ArrMap|)(|f|) =
    L-10-5-N α β 𝔗 𝔎 c(|ArrMap|)(|f|) ∘_A cat-Set β ψ(|NTMap|)(|a|)
  if f : a ↦_op-cat 𝔄 b for a b f
proof–

  note f = that[unfolded cat-op-simps]
  from f have a: a ∈₀ 𝔄(|Obj|) and b: b ∈₀ 𝔄(|Obj|) by auto
```

```
interpret p-a-ε:
  is-cat-rKe-preserves α 𝔅 ℭ 𝔄 ‹cat-Set α› 𝔎 𝔗 𝔊 ‹?H-𝔄 a› ε
  by (rule cat-pw-rKe-preserved[OF a])
interpret a-ε: is-cat-rKe
  α 𝔅 ℭ ‹cat-Set α› 𝔎 ‹?H-𝔄𝔗 a› ‹?H-𝔄𝔊 a› ‹?H-𝔄ε a›
  by (rule p-a-ε.cat-rKe-preserves)
interpret ntcf-ua-fo-a-ε: is-iso-ntcf
  β ?ua-NTDGDom ‹cat-Set β› ‹?ua-NTDom a› ‹?ua-NTCod a› ‹?ua a›
  by (rule a-ε.cat-rKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit[OF β αβ])

interpret p-b-ε:
  is-cat-rKe-preserves α 𝔅 ℭ 𝔄 ‹cat-Set α› 𝔎 𝔗 𝔊 ‹?H-𝔄 b› ε
  by (rule cat-pw-rKe-preserved[OF b])
interpret b-ε: is-cat-rKe
  α 𝔅 ℭ ‹cat-Set α› 𝔎 ‹?H-𝔄𝔗 b› ‹?H-𝔄𝔊 b› ‹?H-𝔄ε b›
  by (rule p-b-ε.cat-rKe-preserves)
interpret ntcf-ua-fo-b-ε: is-iso-ntcf
  β ?ua-NTDGDom ‹cat-Set β› ‹?ua-NTDom b› ‹?ua-NTCod b› ‹?ua b›
  by (rule b-ε.cat-rKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit[OF β αβ])

interpret 𝔎-SET: is-tiny-functor β ‹?FUNCT ℭ› ‹?FUNCT 𝔅› ?SET-𝔎
  by
    (
      rule exp-cat-cf-is-tiny-functor[
        OF β αβ AG.category-cat-Set AG.is-functor-axioms
        ]
    )
from f interpret Hom-f:
  is-ntcf α 𝔄 ‹cat-Set α› ‹?H-𝔄 a› ‹?H-𝔄 b› ‹?H-A f›
  by (cs-concl cs-intro: cat-cs-intros)

let ?cf-hom-lhs =
  ‹
    cf-hom
      (?FUNCT ℭ)
      [ntcf-arrow (ntcf-id (?H-ℭ c)), ntcf-arrow (?H-A𝔊 f)]∘
  ›
let ?cf-hom-rhs =
  ‹
    cf-hom
      (?FUNCT 𝔅)
      [
        ntcf-arrow (ntcf-id (?H-ℭ c ∘CF 𝔎)),
        ntcf-arrow (?H-A f ∘NTCF-CF 𝔗)
      ]∘
  ›
let ?dom =
  ‹Hom (?FUNCT ℭ) (cf-map (?H-ℭ c)) (cf-map (?H-𝔄𝔊 a))›
let ?cod = ‹Hom (?FUNCT 𝔅) (cf-map (?H-ℭ𝔎 c)) (cf-map (?H-𝔄𝔗 b))›
let ?cf-hom-lhs-umap-fo-inter =
  ‹Hom (?FUNCT ℭ) (cf-map (?H-ℭ c)) (cf-map (?H-𝔄𝔊 b))›
let ?umap-fo-cf-hom-rhs-inter =
  ‹Hom (?FUNCT 𝔅) (cf-map (?H-ℭ𝔎 c)) (cf-map (?H-𝔄𝔗 a))›

have [cat-cs-simps]:
  ?umap-fo b ∘A cat-Set β ?cf-hom-lhs =
    ?cf-hom-rhs ∘A cat-Set β ?umap-fo a
```

proof–

from *f assms*(*3*) *αβ* **have** *cf-hom-lhs*:
?cf-hom-lhs : *?dom* $\mapsto_{cat\text{-}Set\ \beta}$ *?cf-hom-lhs-umap-fo-inter*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro:
cat-cs-intros
cat-FUNCT-cs-intros
cat-prod-cs-intros
cat-op-intros
)
from *f assms*(*3*) *αβ* **have** *umap-fo-b*:
?umap-fo b : *?cf-hom-lhs-umap-fo-inter* $\mapsto_{cat\text{-}Set\ \beta}$ *?cod*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro:
cat-cs-intros
cat-FUNCT-cs-intros
cat-prod-cs-intros
cat-op-intros
)
from *cf-hom-lhs umap-fo-b* **have** *umap-fo-cf-hom-lhs*:
?umap-fo b $\circ_{A\ cat\text{-}Set\ \beta}$ *?cf-hom-lhs* : *?dom* $\mapsto_{cat\text{-}Set\ \beta}$ *?cod*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros*
)
then have *dom-umap-fo-cf-hom-lhs*:
$\mathcal{D}_\circ$ ((*?umap-fo b* $\circ_{A\ cat\text{-}Set\ \beta}$ *?cf-hom-lhs*)(|*ArrVal*|)) = *?dom*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros*
)

from *f assms*(*3*) *αβ* **have** *cf-hom-rhs*:
?cf-hom-rhs : *?umap-fo-cf-hom-rhs-inter* $\mapsto_{cat\text{-}Set\ \beta}$ *?cod*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro:
cat-cs-intros
cat-FUNCT-cs-intros
cat-prod-cs-intros
cat-op-intros
)
from *f assms*(*3*) *αβ* **have** *umap-fo-a*:
?umap-fo a : *?dom* $\mapsto_{cat\text{-}Set\ \beta}$ *?umap-fo-cf-hom-rhs-inter*
by
(
cs-concl

```
      cs-simp: cat-cs-simps cat-FUNCT-cs-simps
      cs-intro:
        cat-cs-intros
        cat-FUNCT-cs-intros
        cat-prod-cs-intros
        cat-op-intros
    )
from cf-hom-rhs umap-fo-a have cf-hom-rhs-umap-fo-a:
  ?cf-hom-rhs ∘_A cat-Set β ?umap-fo a : ?dom ↦_cat-Set β ?cod
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros
    )
then have dom-cf-hom-rhs-umap-fo-a:
  𝒟∘ ((?cf-hom-rhs ∘_A cat-Set β ?umap-fo a)(|ArrVal|)) = ?dom
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros
    )

show ?thesis
proof(rule arr-Set-eqI)

  from umap-fo-cf-hom-lhs show arr-Set-umap-fo-cf-hom-lhs:
    arr-Set β (?umap-fo b ∘_A cat-Set β ?cf-hom-lhs)
    by (auto dest: cat-Set-is-arrD(1))
  from cf-hom-rhs-umap-fo-a show arr-Set-cf-hom-rhs-umap-fo-a:
    arr-Set β (?cf-hom-rhs ∘_A cat-Set β ?umap-fo a)
    by (auto dest: cat-Set-is-arrD(1))

  show
    (?umap-fo b ∘_A cat-Set β ?cf-hom-lhs)(|ArrVal|) =
      (?cf-hom-rhs ∘_A cat-Set β ?umap-fo a)(|ArrVal|)
  proof
    (
      rule vsv-eqI,
      unfold
        dom-umap-fo-cf-hom-lhs dom-cf-hom-rhs-umap-fo-a in-Hom-iff;
      (rule refl)?
    )

    fix ℌ assume prems:
      ℌ : cf-map (?H-ℭ c) ↦_?FUNCT ℭ cf-map (?H-𝔄𝔊 a)

    let ?ℌ = ‹ntcf-of-ntcf-arrow ℭ (cat-Set α) ℌ›
    let ?lhs = ‹?H-𝔄ε b ∙_NTCF ((?H-A𝔊 f ∙_NTCF ?ℌ) ∘_NTCF-CF ℜ)›
    let ?rhs =
      ‹(?H-A f ∘_NTCF-CF 𝔗 ∙_NTCF ?H-𝔄ε a ∙_NTCF (?ℌ ∘_NTCF-CF ℜ))›
    let ?cf-hom-𝔄ε = ‹λb b'. cf-hom 𝔄 [𝔄(|CId|)(|b|), ε(|NTMap|)(|b'|)]∘›
    let ?Yc = ‹λQ. Yoneda-component (?H-𝔄 b) a f Q›
    let ?ℌℜ = ‹λb'. ?ℌ(|NTMap|)(|ℜ(|ObjMap|)(|b'|)|)›
    let ?𝔊ℜ = ‹λb'. 𝔊(|ObjMap|)(|ℜ(|ObjMap|)(|b'|)|)›

    have [cat-cs-simps]:
      cf-of-cf-map ℭ (cat-Set α) (cf-map (?H-ℭ c)) = ?H-ℭ c
      by
```

```
    (
      cs-concl cs-shallow
        cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
    )
  have [cat-cs-simps]:
    cf-of-cf-map ℭ (cat-Set α) (cf-map (?H-𝔄𝔊 a)) = ?H-𝔄𝔊 a
    by
    (
      cs-concl cs-shallow
        cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
    )
  note ℌ = cat-FUNCT-is-arrD[OF prems, unfolded cat-cs-simps]
  have Hom-c: ?H-ℭℜ c : 𝔅 ↦↦_{Cα} cat-Set α
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  have [cat-cs-simps]: ?lhs = ?rhs
  proof(rule ntcf-eqI)
    from ℌ(1) f show lhs:
      ?lhs : ?H-ℭℜ c ↦_{CF} ?H-𝔄𝔗 b : 𝔅 ↦↦_{Cα} cat-Set α
      by (cs-concl cs-simp: cs-intro: cat-cs-intros)
    then have dom-lhs: 𝒟∘ (?lhs⦇NTMap⦈) = 𝔅⦇Obj⦈
      by (cs-concl cs-simp: cat-cs-simps)+
    from ℌ(1) f show rhs:
      ?rhs : ?H-ℭℜ c ↦_{CF} ?H-𝔄𝔗 b : 𝔅 ↦↦_{Cα} cat-Set α
      by (cs-concl cs-intro: cat-cs-intros)
    then have dom-rhs: 𝒟∘ (?rhs⦇NTMap⦈) = 𝔅⦇Obj⦈
      by (cs-concl cs-simp: cat-cs-simps)+
    have [cat-cs-simps]:
      ?cf-hom-𝔄ε b b' ∘_{A cat-Set α}
        (?Yc (?𝔊ℜ b') ∘_{A cat-Set α} ?ℌℜ b') =
          ?Yc (𝔗⦇ObjMap⦈⦇b'⦈) ∘_{A cat-Set α}
            (?cf-hom-𝔄ε a b' ∘_{A cat-Set α} ?ℌℜ b')
      (is ‹?lhs-Set = ?rhs-Set›)
      if b' ∈_∘ 𝔅⦇Obj⦈ for b'
    proof–
      let ?ℜb' = ‹ℜ⦇ObjMap⦈⦇b'⦈›
      from ℌ(1) f that assms(3) Ran.HomCod.category-axioms
      have lhs-Set-is-arr: ?lhs-Set :
        Hom ℭ c (?ℜb') ↦_{cat-Set α} Hom 𝔄 b (𝔗⦇ObjMap⦈⦇b'⦈)
        by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro:
              cat-cs-intros cat-prod-cs-intros cat-op-intros
        )
      then have dom-lhs-Set: 𝒟∘ (?lhs-Set⦇ArrVal⦈) = Hom ℭ c ?ℜb'
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      from ℌ(1) f that assms(3) Ran.HomCod.category-axioms
      have rhs-Set-is-arr: ?rhs-Set :
        Hom ℭ c (?ℜb') ↦_{cat-Set α} Hom 𝔄 b (𝔗⦇ObjMap⦈⦇b'⦈)
        by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro:
              cat-cs-intros cat-prod-cs-intros cat-op-intros
        )
```

```
      then have dom-rhs-Set: 𝒟∘ (?rhs-Set(|ArrVal|)) = Hom ℭ c ?ℜb′
        by (cs-concl cs-shallow cs-simp: cat-cs-simps)
      show ?thesis
      proof(rule arr-Set-eqI)
        from lhs-Set-is-arr show arr-Set α ?lhs-Set
          by (auto dest: cat-Set-is-arrD(1))
        from rhs-Set-is-arr show arr-Set α ?rhs-Set
          by (auto dest: cat-Set-is-arrD(1))
        show ?lhs-Set(|ArrVal|) = ?rhs-Set(|ArrVal|)
        proof(rule vsv-eqI, unfold dom-lhs-Set dom-rhs-Set in-Hom-iff)
          fix h assume h : c ↦ℭ ?ℜb′
          with ℌ(1) f that assms Ran.HomCod.category-axioms show
            ?lhs-Set(|ArrVal|)(|h|) = ?rhs-Set(|ArrVal|)(|h|)
            by
              (
                cs-concl
                  cs-simp: cat-cs-simps cat-op-simps
                  cs-intro:
                    cat-cs-intros cat-prod-cs-intros cat-op-intros
              )
        qed (use arr-Set-lhs-Set arr-Set-rhs-Set in auto)
      qed
        (
          use lhs-Set-is-arr rhs-Set-is-arr in
            ‹cs-concl cs-shallow cs-simp: cat-cs-simps›
        )+

  qed

  show ?lhs(|NTMap|) = ?rhs(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix b′ assume b′ ∈∘ 𝔅(|Obj|)
    with ℌ(1) f assms(3) show ?lhs(|NTMap|)(|b′|) = ?rhs(|NTMap|)(|b′|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros)

qed simp-all

from
  assms(3) f ℌ(1) prems αβ

  Ran.HomCod.category-axioms
  FUNCT-ℭ.category-axioms
  FUNCT-𝔅.category-axioms
  AG.is-functor-axioms
  Ran.is-functor-axioms
  Hom-f.is-ntcf-axioms
show
  (?umap-fo b ∘A cat-Set β ?cf-hom-lhs)(|ArrVal|)(|ℌ|) =
    (?cf-hom-rhs ∘A cat-Set β ?umap-fo a)(|ArrVal|)(|ℌ|)
    by (subst (1 2) ℌ(2))
      (
        cs-concl
```

```
                cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
                cs-intro:
                  cat-cs-intros
                  cat-prod-cs-intros
                  cat-FUNCT-cs-intros
                  cat-op-intros
              )
          qed
            (
              use arr-Set-umap-fo-cf-hom-lhs arr-Set-cf-hom-rhs-umap-fo-a in
                auto
            )

        qed
          (
            use umap-fo-cf-hom-lhs cf-hom-rhs-umap-fo-a in
              ‹cs-concl cs-shallow cs-simp: cat-cs-simps›
          )+

      qed

      from f assms αβ show ?thesis
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-Kan-cs-simps cat-FUNCT-cs-simps
              cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )

    qed

  qed auto

qed
```

```
from L-10-5-χ-is-iso-ntcf[OF β αβ assms] have inv-χ:
  inv-ntcf (L-10-5-χ α β 𝔗 ℜ c) :
    L-10-5-N α β 𝔗 ℜ c ↦_CF.iso cf-Cone α β ?𝔗-cℜ :
    op-cat 𝔄 ↦↦_Cβ cat-Set β
  by (auto intro: iso-ntcf-is-iso-arr)

define φ where φ = inv-ntcf (L-10-5-χ α β 𝔗 ℜ c) ∘_NTCF ψ ∘_NTCF inv-ntcf Y'

from inv-Y' ψ inv-χ have φ: φ :
  Hom_O.Cβ𝔄(−,?𝔊c) ↦_CF.iso cf-Cone α β ?𝔗-cℜ :
  op-cat 𝔄 ↦↦_Cβ cat-Set β
  unfolding φ-def by (cs-concl cs-shallow cs-intro: cat-cs-intros)

interpret φ: is-iso-ntcf
  β ‹op-cat 𝔄› ‹cat-Set β› ‹Hom_O.Cβ𝔄(−,?𝔊c)› ‹cf-Cone α β ?𝔗-cℜ› φ
  by (rule φ)

let ?φ-𝔊c-CId = ‹φ⟦NTMap⟧⟦?𝔊c⟧⟦ArrVal⟧⟦𝔄⟦CId⟧⟦?𝔊c⟧⟧›
let ?ntcf-φ-𝔊c-CId = ‹ntcf-of-ntcf-arrow (c ↓_CF ℜ) 𝔄 ?φ-𝔊c-CId›
```

from *AG.vempty-is-zet assms*(*3*) **have** Δ: $?\Delta : \mathfrak{A} \mapsto\mapsto_{C\beta} ?c\mathfrak{K}\text{-}\mathfrak{A}$
  by
    (
      cs-concl **cs-shallow**
        cs-simp: *cat-comma-cs-simps*
        cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
from *assms*(*3*) **have** $\mathfrak{G}c$: $?\mathfrak{G}c \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *AG.vempty-is-zet* **have** $\mathfrak{T}$-$c\mathfrak{K}$: *cf-map* ($?\mathfrak{T}$-$c\mathfrak{K}$) $\in_{\circ}$ $?c\mathfrak{K}\text{-}\mathfrak{A}(\!|Obj|\!)$
  by
    (
      cs-concl
        cs-simp: *cat-FUNCT-components*(*1*)
        cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)

from
  φ*.ntcf-NTMap-is-arr*[*unfolded cat-op-simps*, *OF* $\mathfrak{G}c$]
  assms(*3*)
  AG.vempty-is-zet
  β*.vempty-is-zet*
  $\alpha\beta$
have φ-$\mathfrak{G}c$: $\varphi(\!|NTMap|\!)(\!|?\mathfrak{G}c|\!)$:
  Hom $\mathfrak{A}$ $?\mathfrak{G}c ?\mathfrak{G}c \mapsto_{cat\text{-}Set\ \beta}$
  Hom $?c\mathfrak{K}\text{-}\mathfrak{A}$ (*cf-map* ($?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A}$ $?\mathfrak{G}c$)) (*cf-map* $?\mathfrak{T}$-$c\mathfrak{K}$)
  by
    (
      cs-prems
        cs-simp:
          cat-cs-simps
          cat-Kan-cs-simps
          cat-comma-cs-simps
          cat-op-simps
          cat-FUNCT-components(*1*)
        cs-intro:
          cat-Kan-cs-intros
          cat-comma-cs-intros
          cat-cs-intros
          cat-FUNCT-cs-intros
          cat-op-intros
)

with *assms*(*3*) **have** φ-$\mathfrak{G}c$-*CId*:
  $?\varphi$-$\mathfrak{G}c$-*CId* : *cf-map* ($?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A}$ $?\mathfrak{G}c$) $\mapsto_{?c\mathfrak{K}\text{-}\mathfrak{A}}$ *cf-map* $?\mathfrak{T}$-$c\mathfrak{K}$
  by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
have *ntcf-arrow-*φ-$\mathfrak{G}c$-*CId*: *ntcf-arrow* $?ntcf\text{-}\varphi\text{-}\mathfrak{G}c\text{-}CId$ = $?\varphi$-$\mathfrak{G}c$-*CId*
  by (*rule cat-FUNCT-is-arrD*(*2*)[*OF* φ-$\mathfrak{G}c$-*CId*, *symmetric*])

have *ua*: *universal-arrow-fo* $?\Delta$ (*cf-map* ($?\mathfrak{T}$-$c\mathfrak{K}$)) $?\mathfrak{G}c$ $?\varphi$-$\mathfrak{G}c$-*CId*
  by
    (
      rule is-functor.cf-universal-arrow-fo-if-is-iso-ntcf[
        OF Δ $\mathfrak{G}c$ $\mathfrak{T}$-$c\mathfrak{K}$ φ[*unfolded cf-Cone-def cat-cs-simps*]
        ]
)
moreover have *ntcf-*φ-$\mathfrak{G}c$-*CId*:
  $?ntcf\text{-}\varphi\text{-}\mathfrak{G}c\text{-}CId$: $?\mathfrak{G}c <_{CF.cone} ?\mathfrak{T}$-$c\mathfrak{K}$: $c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

proof(*intro is-cat-coneI*)
from *cat-FUNCT-is-arrD*(*1*)[*OF* φ-$\mathfrak{G}c$-*CId*] *assms*(*3*) *AG.vempty-is-zet* **show**
ntcf-of-ntcf-arrow (c $\downarrow_{CF}$ $\mathfrak{K}$) $\mathfrak{A}$ *?*φ-$\mathfrak{G}c$-*CId* :
?cf-c$\mathfrak{K}$-$\mathfrak{A}$ *?*$\mathfrak{G}c$ $\mapsto_{CF}$ *?*$\mathfrak{T}$-*c*$\mathfrak{K}$: c $\downarrow_{CF}$ $\mathfrak{K}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
by
(
cs-prems
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)
qed (*rule* $\mathfrak{G}c$)
ultimately have *?ntcf-*φ-$\mathfrak{G}c$-*CId* : *?*$\mathfrak{G}c$ $<_{CF.lim}$ *?*$\mathfrak{T}$-*c*$\mathfrak{K}$: c $\downarrow_{CF}$ $\mathfrak{K}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
by (*intro is-cat-cone.cat-cone-is-cat-limit*)
(*simp-all add*: *ntcf-arrow-*φ-$\mathfrak{G}c$-*CId*)
then show *?thesis* **using** *that* **by** *auto*

qed

lemma (**in** *is-cat-pw-lKe*) *cat-pw-lKe-ex-cat-colimit*:
— Based on the elements of Chapter X-5 in [9].
assumes $\mathfrak{K}$: $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
and $\mathfrak{T}$: $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
and c $\in_{\circ}$ $\mathfrak{C}(\!|Obj|\!)$
obtains *UA*
where *UA* : $\mathfrak{T}$ $\circ_{CF}$ $\mathfrak{K}$ $_{CF}\bigsqcap_{O}$ c $>_{CF.colim}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)$: $\mathfrak{K}$ $_{CF}\downarrow$ c $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
proof–
from
is-cat-pw-rKe.cat-pw-rKe-ex-cat-limit
[
OF is-cat-pw-rKe-op AG.is-functor-op ntcf-lKe.NTDom.is-functor-op,
unfolded cat-op-simps,
OF assms(*3*)
]
obtain *UA* **where** *UA*: *UA* :
$\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)$ $<_{CF.lim}$ *op-cf* $\mathfrak{T}$ $\circ_{CF}$ c $_{O}\bigsqcap_{CF}$ (*op-cf* $\mathfrak{K}$) :
c $\downarrow_{CF}$ (*op-cf* $\mathfrak{K}$) $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{A}$
by *auto*
from *assms*(*3*) **have** [*cat-cs-simps*]:
op-cf $\mathfrak{T}$ $\circ_{CF}$ c $_{O}\bigsqcap_{CF}$ (*op-cf* $\mathfrak{K}$) $\circ_{CF}$ *op-cf-obj-comma* $\mathfrak{K}$ c =
op-cf $\mathfrak{T}$ $\circ_{CF}$ *op-cf* ($\mathfrak{K}$ $_{CF}\bigsqcap_{O}$ c)
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps AG.op-cf-cf-obj-comma-proj*[*OF assms*(*3*)]
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
from *assms*(*3*) **have** [*cat-op-simps*]:
$\mathfrak{T}$ $\circ_{CF}$ *op-cf* (c $_{O}\bigsqcap_{CF}$ (*op-cf* $\mathfrak{K}$)) $\circ_{CF}$ *op-cf* (*op-cf-obj-comma* $\mathfrak{K}$ c) =
$\mathfrak{T}$ $\circ_{CF}$ $\mathfrak{K}$ $_{CF}\bigsqcap_{O}$ c
by
(
cs-concl **cs-shallow**
cs-simp:
cat-cs-simps cat-op-simps
op-cf-cf-comp[*symmetric*] *AG.op-cf-cf-obj-comma-proj*[*symmetric*]
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
from *assms*(*3*) **have** [*cat-op-simps*]: *op-cat* (*op-cat* ($\mathfrak{K}$ $_{CF}\downarrow$ c)) = $\mathfrak{K}$ $_{CF}\downarrow$ c

```
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  note ntcf-cf-comp-is-cat-limit-if-is-iso-functor =
    ntcf-cf-comp-is-cat-limit-if-is-iso-functor
      [
        OF UA AG.op-cf-obj-comma-is-iso-functor[OF assms(3)],
        unfolded cat-op-simps
      ]
  have op-ntcf UA ∘_{NTCF-CF} op-cf (op-cf-obj-comma ℜ c) :
    𝔗 ∘_{CF} ℜ _{CF}↑_O c >_{CF.colim} 𝔉(|ObjMap|)(|c|) : ℜ _{CF}↓ c ↦↦_{Cα} 𝔄
    by
      (
        rule is-cat-limit.is-cat-colimit-op
          [
            OF ntcf-cf-comp-is-cat-limit-if-is-iso-functor,
            unfolded cat-op-simps
          ]
      )
  then show ?thesis using that by auto
qed
```

15.10 The limit and the colimit for the pointwise Kan extensions

15.10.1 Definition and elementary properties

See Theorem 3 in Chapter X-5 in [9].

```
definition the-pw-cat-rKe-limit :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where the-pw-cat-rKe-limit α ℜ 𝔗 𝔊 c =
    [
      𝔊(|ObjMap|)(|c|),
      (
        SOME UA.
          UA : 𝔊(|ObjMap|)(|c|) <_{CF.lim} 𝔗 ∘_{CF} c _{O}↑_{CF} ℜ : c ↓_{CF} ℜ ↦↦_{Cα} 𝔗(|HomCod|)
      )
    ]_∘

definition the-pw-cat-lKe-colimit :: V ⇒ V ⇒ V ⇒ V ⇒ V ⇒ V
  where the-pw-cat-lKe-colimit α ℜ 𝔗 𝔉 c =
    [
      𝔉(|ObjMap|)(|c|),
      op-ntcf
        (
          the-pw-cat-rKe-limit α (op-cf ℜ) (op-cf 𝔗) (op-cf 𝔉) c(|UArr|) ∘_{NTCF-CF}
          op-cf-obj-comma ℜ c
        )
    ]_∘
```

Components.

```
lemma the-pw-cat-rKe-limit-components:
  shows the-pw-cat-rKe-limit α ℜ 𝔗 𝔊 c(|UObj|) = 𝔊(|ObjMap|)(|c|)
    and the-pw-cat-rKe-limit α ℜ 𝔗 𝔊 c(|UArr|) =
      (
        SOME UA.
          UA : 𝔊(|ObjMap|)(|c|) <_{CF.lim} 𝔗 ∘_{CF} c _{O}↑_{CF} ℜ : c ↓_{CF} ℜ ↦↦_{Cα} 𝔗(|HomCod|)
      )
```

unfolding *the-pw-cat-rKe-limit-def ua-field-simps*
by (*simp-all add*: *nat-omega-simps*)

lemma *the-pw-cat-lKe-colimit-components*:
shows *the-pw-cat-lKe-colimit* α $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ $c(\!|UObj|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)$
and *the-pw-cat-lKe-colimit* α $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ $c(\!|UArr|\!) = op\text{-}ntcf$
(
the-pw-cat-rKe-limit α (*op-cf* $\mathfrak{K}$) (*op-cf* $\mathfrak{T}$) (*op-cf* $\mathfrak{F}$) $c(\!|UArr|\!) \circ_{NTCF-CF}$
op-cf-obj-comma $\mathfrak{K}$ c
)
unfolding *the-pw-cat-lKe-colimit-def ua-field-simps*
by (*simp-all add*: *nat-omega-simps*)

context *is-functor*
begin

lemmas *the-pw-cat-rKe-limit-components'* =
the-pw-cat-rKe-limit-components[**where** $\mathfrak{T}$=$\mathfrak{F}$, *unfolded cat-cs-simps*]

end

15.10.2 The limit for the pointwise right Kan extension is a limit, the colimit for the pointwise left Kan extension is a colimit

lemma (**in** *is-cat-pw-rKe*) *cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$ **and** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *the-pw-cat-rKe-limit* α $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ $c(\!|UArr|\!)$:
the-pw-cat-rKe-limit α $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{G}$ $c(\!|UObj|\!) <_{CF.lim} \mathfrak{T} \circ_{CF} c\ {}_O\!\sqcap_{CF} \mathfrak{K}$:
$c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–
from *cat-pw-rKe-ex-cat-limit*[*OF assms*] **obtain** *UA*
where *UA*: $UA : \mathfrak{G}(\!|ObjMap|\!)(\!|c|\!) <_{CF.lim} \mathfrak{T} \circ_{CF} c\ {}_O\!\sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by *auto*
show *?thesis*
unfolding *the-pw-cat-rKe-limit-components*
by (*rule someI2*, *unfold cat-cs-simps*, *rule UA*)
qed

lemma (**in** *is-cat-pw-lKe*) *cat-pw-lKe-the-pw-cat-lKe-colimit-is-cat-colimit*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$ **and** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *the-pw-cat-lKe-colimit* α $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ $c(\!|UArr|\!)$:
$\mathfrak{T} \circ_{CF} \mathfrak{K}\ {}_{CF}\!\sqcap_O c >_{CF.colim}$ *the-pw-cat-lKe-colimit* α $\mathfrak{K}$ $\mathfrak{T}$ $\mathfrak{F}$ $c(\!|UObj|\!)$:
$\mathfrak{K}\ {}_{CF}\!\downarrow c \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–
interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{T}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*2*))
note *cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit* =
is-cat-pw-rKe.cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit
[
OF is-cat-pw-rKe-op AG.is-functor-op ntcf-lKe.NTDom.is-functor-op,
unfolded cat-op-simps,
OF assms(*3*)
]
from *assms*(*3*) **have**
op-cf $\mathfrak{T} \circ_{CF} c\ {}_O\!\sqcap_{CF}$ (*op-cf* $\mathfrak{K}$) $\circ_{CF}$ *op-cf-obj-comma* $\mathfrak{K}$ c =
op-cf $\mathfrak{T} \circ_{CF}$ *op-cf* ($\mathfrak{K}\ {}_{CF}\!\sqcap_O c$)
by
(

```
      cs-concl cs-shallow
        cs-simp:
          cat-cs-simps cat-comma-cs-simps cat-op-simps
          AG.op-cf-cf-obj-comma-proj[OF assms(3)]
        cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
    )
  note ntcf-cf-comp-is-cat-limit-if-is-iso-functor =
    ntcf-cf-comp-is-cat-limit-if-is-iso-functor
      [
        OF
          cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit
          AG.op-cf-obj-comma-is-iso-functor[OF assms(3)],
          unfolded this, folded op-cf-cf-comp
      ]
  from assms(3) have [cat-op-simps]: op-cat (op-cat (𝔎 CF↓ c)) = 𝔎 CF↓ c
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  from assms(3) have [cat-op-simps]: op-cf (op-cf (𝔎 CF↑ O c)) = 𝔎 CF↑ O c
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  have [cat-op-simps]:
    the-pw-cat-rKe-limit α (op-cf 𝔎) (op-cf 𝔗) (op-cf 𝔉) c(|UObj|) =
      the-pw-cat-lKe-colimit α 𝔎 𝔗 𝔉 c(|UObj|)
    unfolding
      the-pw-cat-lKe-colimit-components
      the-pw-cat-rKe-limit-components
      cat-op-simps
    by simp
  show ?thesis
    by
      (
        rule is-cat-limit.is-cat-colimit-op
          [
            OF ntcf-cf-comp-is-cat-limit-if-is-iso-functor,
            folded the-pw-cat-lKe-colimit-components,
            unfolded cat-op-simps
          ]
      )
qed

lemma (in is-cat-pw-rKe) cat-pw-rKe-the-ntcf-rKe-is-cat-rKe:
  assumes 𝔎 : 𝔅 ↦↦Cα ℭ and 𝔗 : 𝔅 ↦↦Cα 𝔄
  shows the-ntcf-rKe α 𝔗 𝔎 (the-pw-cat-rKe-limit α 𝔎 𝔗 𝔊) :
    the-cf-rKe α 𝔗 𝔎 (the-pw-cat-rKe-limit α 𝔎 𝔗 𝔊) ∘CF 𝔎 ↦CF.rKeα 𝔗 :
    𝔅 ↦C ℭ ↦C 𝔄
proof–
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  show the-ntcf-rKe α 𝔗 𝔎 (the-pw-cat-rKe-limit α 𝔎 𝔗 𝔊) :
    the-cf-rKe α 𝔗 𝔎 (the-pw-cat-rKe-limit α 𝔎 𝔗 𝔊) ∘CF 𝔎 ↦CF.rKeα 𝔗 :
    𝔅 ↦C ℭ ↦C 𝔄
    by
      (
```

```
      rule
        the-ntcf-rKe-is-cat-rKe
          [
            OF
              assms(1)
              ntcf-rKe.NTCod.is-functor-axioms
              cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit[OF assms]
          ]
    )
qed

lemma (in is-cat-pw-lKe) cat-pw-lKe-the-ntcf-lKe-is-cat-lKe:
  assumes 𝔎 : 𝔅 ↦↦_{Cα} ℭ and 𝔗 : 𝔅 ↦↦_{Cα} 𝔄
  shows the-ntcf-lKe α 𝔗 𝔎 (the-pw-cat-lKe-colimit α 𝔎 𝔗 𝔉) :
    𝔗 ↦_{CF.lKeα} the-cf-lKe α 𝔗 𝔎 (the-pw-cat-lKe-colimit α 𝔎 𝔗 𝔉) ∘_{CF} 𝔎 :
    𝔅 ↦_C ℭ ↦_C 𝔄
proof–
  interpret 𝔗: is-functor α 𝔅 𝔄 𝔗 by (rule assms(2))
  show ?thesis
    by
      (
        rule the-ntcf-lKe-is-cat-lKe
          [
            OF
              assms(1,2)
              cat-pw-lKe-the-pw-cat-lKe-colimit-is-cat-colimit[OF assms],
            simplified
          ]
      )
qed
```

16 Pointwise Kan extensions: application example

16.1 Background

The application example presented in this section is based on Exercise 6.1.ii in [14]. The primary purpose of this section is the instantiation of the locales associated with the pointwise Kan extensions.

lemma *cat-ordinal-2-is-arrE*:
assumes $f : a \mapsto_{cat\text{-}ordinal\ (2_{\mathbb{N}})} b$
obtains $f = [0, 0]_{\circ}$ **and** $a = 0$ **and** $b = 0$
| $f = [0, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 0$ **and** $b = 1_{\mathbb{N}}$
| $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 1_{\mathbb{N}}$ **and** $b = 1_{\mathbb{N}}$
using *cat-ordinal-is-arrD*[*OF assms*] **unfolding** *two* **by** *auto*

lemma *cat-ordinal-3-is-arrE*:
assumes $f : a \mapsto_{cat\text{-}ordinal\ (3_{\mathbb{N}})} b$
obtains $f = [0, 0]_{\circ}$ **and** $a = 0$ **and** $b = 0$
| $f = [0, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 0$ **and** $b = 1_{\mathbb{N}}$
| $f = [0, 2_{\mathbb{N}}]_{\circ}$ **and** $a = 0$ **and** $b = 2_{\mathbb{N}}$
| $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 1_{\mathbb{N}}$ **and** $b = 1_{\mathbb{N}}$
| $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$ **and** $a = 1_{\mathbb{N}}$ **and** $b = 2_{\mathbb{N}}$
| $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$ **and** $a = 2_{\mathbb{N}}$ **and** $b = 2_{\mathbb{N}}$
using *cat-ordinal-is-arrD*[*OF assms*] **unfolding** *three* **by** *auto*

lemma *0123*: $0 \in_{\circ} 2_{\mathbb{N}}$ $1_{\mathbb{N}} \in_{\circ} 2_{\mathbb{N}}$ $0 \in_{\circ} 3_{\mathbb{N}}$ $1_{\mathbb{N}} \in_{\circ} 3_{\mathbb{N}}$ $2_{\mathbb{N}} \in_{\circ} 3_{\mathbb{N}}$ **by** *auto*

16.2 $\mathfrak{K}23$

16.2.1 Definition and elementary properties

definition $\mathfrak{K}23 :: V$
where $\mathfrak{K}23 =$
[
($\lambda a \in_{\circ} cat\text{-}ordinal\ (2_{\mathbb{N}})(\!|Obj|\!).$ *if* $a = 0$ *then* 0 *else* $2_{\mathbb{N}}$),
(
$\lambda f \in_{\circ} cat\text{-}ordinal\ (2_{\mathbb{N}})(\!|Arr|\!).$
if $f = [0, 0]_{\circ} \Rightarrow [0, 0]_{\circ}$
| $f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow [0, 2_{\mathbb{N}}]_{\circ}$
| $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
| *otherwise* $\Rightarrow 0$
),
cat-ordinal $(2_{\mathbb{N}})$,
cat-ordinal $(3_{\mathbb{N}})$
]$_{\circ}$

Components.

lemma $\mathfrak{K}23$-*components*:
shows $\mathfrak{K}23(\!|ObjMap|\!) = (\lambda a \in_{\circ} cat\text{-}ordinal\ (2_{\mathbb{N}})(\!|Obj|\!).$ *if* $a = 0$ *then* 0 *else* $2_{\mathbb{N}})$
and $\mathfrak{K}23(\!|ArrMap|\!) =$
(
$\lambda f \in_{\circ} cat\text{-}ordinal\ (2_{\mathbb{N}})(\!|Arr|\!).$
if $f = [0, 0]_{\circ} \Rightarrow [0, 0]_{\circ}$
| $f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow [0, 2_{\mathbb{N}}]_{\circ}$
| $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
| *otherwise* $\Rightarrow 0$
)

and [*cat-Kan-cs-simps*]: $\mathfrak{K}23(\lfloor HomDom \rfloor) = cat\text{-}ordinal\ (2_{\mathbb{N}})$
and [*cat-Kan-cs-simps*]: $\mathfrak{K}23(\lfloor HomCod \rfloor) = cat\text{-}ordinal\ (3_{\mathbb{N}})$
unfolding *ℜ23-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

16.2.2 Object map

mk-VLambda *ℜ23-components*(*1*)
|*vsv ℜ23-ObjMap-vsv*[*cat-Kan-cs-intros*]|
|*vdomain ℜ23-ObjMap-vdomain*[*cat-Kan-cs-simps*]|
|*app ℜ23-ObjMap-app*|

lemma *ℜ23-ObjMap-app-0*[*cat-Kan-cs-simps*]:
assumes $x = 0$
shows $\mathfrak{K}23(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) = 0$
by
(
cs-concl
cs-simp: *ℜ23-ObjMap-app cat-ordinal-cs-simps V-cs-simps assms*
cs-intro: *nat-omega-intros*
)

lemma *ℜ23-ObjMap-app-1*[*cat-Kan-cs-simps*]:
assumes $x = 1_{\mathbb{N}}$
shows $\mathfrak{K}23(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) = 2_{\mathbb{N}}$
by
(
cs-concl
cs-simp:
cat-ordinal-cs-simps V-cs-simps omega-of-set ℜ23-ObjMap-app assms
cs-intro: *nat-omega-intros V-cs-intros*
)

16.2.3 Arrow map

mk-VLambda *ℜ23-components*(*2*)
|*vsv ℜ23-ArrMap-vsv*[*cat-Kan-cs-intros*]|
|*vdomain ℜ23-ArrMap-vdomain*[*cat-Kan-cs-simps*]|
|*app ℜ23-ArrMap-app*|

lemma *ℜ23-ArrMap-app-00*[*cat-Kan-cs-simps*]:
assumes $f = [0, 0]_{\circ}$
shows $\mathfrak{K}23(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = [0, 0]_{\circ}$
unfolding *assms*
by
(
cs-concl
cs-simp: *ℜ23-ArrMap-app cat-ordinal-cs-simps V-cs-simps*
cs-intro: *cat-ordinal-cs-intros nat-omega-intros*
)

lemma *ℜ23-ArrMap-app-01*[*cat-Kan-cs-simps*]:
assumes $f = [0, 1_{\mathbb{N}}]_{\circ}$
shows $\mathfrak{K}23(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = [0, 2_{\mathbb{N}}]_{\circ}$
proof–
have $[0, 1_{\mathbb{N}}]_{\circ} \in_{\circ} ordinal\text{-}arrs\ (2_{\mathbb{N}})$
by
(
cs-concl

```
      cs-simp: omega-of-set
      cs-intro: cat-ordinal-cs-intros V-cs-intros nat-omega-intros
    )
  then show ?thesis
    unfolding assms by (simp add: ℜ23-components cat-ordinal-components)
qed

lemma ℜ23-ArrMap-app-11[cat-Kan-cs-simps]:
  assumes f = [1_ℕ, 1_ℕ]_∘
  shows ℜ23(‖ArrMap‖)(‖f‖) = [2_ℕ, 2_ℕ]_∘
proof–
  have [1_ℕ, 1_ℕ]_∘ ∈_∘ ordinal-arrs (2_ℕ)
    by
      (
        cs-concl cs-shallow
          cs-simp: omega-of-set
          cs-intro: cat-ordinal-cs-intros V-cs-intros nat-omega-intros
      )
  then show ?thesis
    unfolding assms by (simp add: ℜ23-components cat-ordinal-components)
qed
```

16.2.4 $\mathfrak{R}23$ is a tiny functor

```
lemma (in Z) ℜ23-is-functor: ℜ23 : cat-ordinal (2_ℕ) ↦↦_Cα cat-ordinal (3_ℕ)
proof–

  from ord-of-nat-ω interpret cat-ordinal-2: finite-category α ‹cat-ordinal (2_ℕ)›
    by (cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros)
  from ord-of-nat-ω interpret cat-ordinal-3: finite-category α ‹cat-ordinal (3_ℕ)›
    by (cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros)

  show ?thesis
  proof(intro is-tiny-functorI' is-functorI')

    show vfsequence ℜ23 unfolding ℜ23-def by auto
    show vcard ℜ23 = 4_ℕ unfolding ℜ23-def by (simp add: nat-omega-simps)

    show ℛ_∘ (ℜ23(‖ObjMap‖)) ⊆_∘ cat-ordinal (3_ℕ)(‖Obj‖)
    proof
      (
        rule vsv.vsv-vrange-vsubset,
        unfold cat-Kan-cs-simps cat-ordinal-cs-simps,
        intro cat-Kan-cs-intros
      )
      fix x assume x ∈_∘ 2_ℕ
      then consider ‹x = 0› | ‹x = 1_ℕ› unfolding two by auto
      then show ℜ23(‖ObjMap‖)(‖x‖) ∈_∘ 3_ℕ
        by (cases, use nothing in ‹simp-all only:›)
          (
            cs-concl
              cs-simp: cat-Kan-cs-simps omega-of-set cs-intro: nat-omega-intros
          )+
    qed

    show ℜ23(‖ArrMap‖)(‖f‖) : ℜ23(‖ObjMap‖)(‖a‖) ↦_cat-ordinal (3_ℕ) ℜ23(‖ObjMap‖)(‖b‖)
      if f : a ↦_cat-ordinal (2_ℕ) b for a b f
      using that
```

```
    by (elim cat-ordinal-2-is-arrE; simp only:)
      (
        cs-concl
          cs-simp: omega-of-set cat-Kan-cs-simps
          cs-intro: nat-omega-intros V-cs-intros cat-ordinal-cs-intros
      )

  show
    ℜ23(|ArrMap|)(|g ∘_A cat-ordinal (2_ℕ) f|) =
      ℜ23(|ArrMap|)(|g|) ∘_A cat-ordinal (3_ℕ) ℜ23(|ArrMap|)(|f|)
    if g : b ↦_cat-ordinal (2_ℕ) c and f : a ↦_cat-ordinal (2_ℕ) b
    for b c g a f
  proof–
    have 0 ∈_∘ 3_ℕ 1_ℕ ∈_∘ 3_ℕ 2_ℕ ∈_∘ 3_ℕ by auto
    then show ?thesis
      using that
      by (elim cat-ordinal-2-is-arrE; simp only:)
        (
          cs-concl
            cs-simp: cat-ordinal-cs-simps cat-Kan-cs-simps
            cs-intro: V-cs-intros cat-ordinal-cs-intros
        )+
  qed

  show
    ℜ23(|ArrMap|)(|cat-ordinal (2_ℕ)(|CId|)(|c|)|) =
      cat-ordinal (3_ℕ)(|CId|)(|ℜ23(|ObjMap|)(|c|)|)
    if c ∈_∘ cat-ordinal (2_ℕ)(|Obj|) for c
  proof–
    from that consider ‹c = 0› | ‹c = 1_ℕ›
      unfolding cat-ordinal-components(1) two by auto
    then show ?thesis
      by (cases, use nothing in ‹simp-all only:›)
        (
          cs-concl
            cs-simp: omega-of-set cat-Kan-cs-simps cat-ordinal-cs-simps
            cs-intro: nat-omega-intros cat-ordinal-cs-intros
        )
  qed

qed (auto intro!: cat-cs-intros simp: ℜ23-components)

qed

lemma (in 𝒵) ℜ23-is-functor'[cat-Kan-cs-intros]:
  assumes 𝔄' = cat-ordinal (2_ℕ)
    and 𝔅' = cat-ordinal (3_ℕ)
  shows ℜ23 : 𝔄' ↦↦_Cα 𝔅'
  unfolding assms by (rule ℜ23-is-functor)

lemmas [cat-Kan-cs-intros] = 𝒵.ℜ23-is-functor'

lemma (in 𝒵) ℜ23-is-tiny-functor:
  ℜ23 : cat-ordinal (2_ℕ) ↦↦_C.tinyα cat-ordinal (3_ℕ)
proof–
  from ord-of-nat-ω interpret cat-ordinal-2: finite-category α ‹cat-ordinal (2_ℕ)›
    by (cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros)
  from ord-of-nat-ω interpret cat-ordinal-3: finite-category α ‹cat-ordinal (3_ℕ)›
```

by (*cs-concl* **cs-shallow cs-intro**: *cat-ordinal-cs-intros*)
show *?thesis*
by (*intro is-tiny-functorI'* $\mathfrak{K}$*23-is-functor*)
(*auto intro*!: *cat-small-cs-intros*)
qed

lemma (in $\mathcal{Z}$**)** $\mathfrak{K}$*23-is-tiny-functor'*[*cat-Kan-cs-intros*]:
assumes $\mathfrak{A}'$ = *cat-ordinal* ($2_{\mathbb{N}}$)
and $\mathfrak{B}'$ = *cat-ordinal* ($3_{\mathbb{N}}$)
shows $\mathfrak{K}23 : \mathfrak{A}' \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule* $\mathfrak{K}$*23-is-tiny-functor*)

lemmas [*cat-Kan-cs-intros*] = $\mathcal{Z}$.$\mathfrak{K}$*23-is-tiny-functor'*

16.3 *LK23*: the functor associated with the left Kan extension along $\mathfrak{K}$*23*

16.3.1 Definition and elementary properties

definition *LK23* :: $V \Rightarrow V$
where *LK23* $\mathfrak{F}$ =
[
(
$\lambda a \in_{\circ}$ *cat-ordinal* ($3_{\mathbb{N}}$)(|*Obj*|).
if $a = 0 \Rightarrow \mathfrak{F}$(|*ObjMap*|)(|0|)
| $a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F}$(|*ObjMap*|)(|0|)
| $a = 2_{\mathbb{N}} \Rightarrow \mathfrak{F}$(|*ObjMap*|)(|$1_{\mathbb{N}}$|)
| *otherwise* $\Rightarrow \mathfrak{F}$(|*HomCod*|)(|*Obj*|)
),
(
$\lambda f \in_{\circ}$ *cat-ordinal* ($3_{\mathbb{N}}$)(|*Arr*|).
if $f = [0, 0]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, 0|)$_{\bullet}$
| $f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, 0|)$_{\bullet}$
| $f = [0, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, $1_{\mathbb{N}}$|)$_{\bullet}$
| $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, 0|)$_{\bullet}$
| $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, $1_{\mathbb{N}}$|)$_{\bullet}$
| $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|$1_{\mathbb{N}}$, $1_{\mathbb{N}}$|)$_{\bullet}$
| *otherwise* $\Rightarrow \mathfrak{F}$(|*HomCod*|)(|*Arr*|)
),
cat-ordinal ($3_{\mathbb{N}}$),
$\mathfrak{F}$(|*HomCod*|)
]$_{\circ}$

Components.

lemma *LK23-components*:
shows *LK23* $\mathfrak{F}$(|*ObjMap*|) =
(
$\lambda a \in_{\circ}$ *cat-ordinal* ($3_{\mathbb{N}}$)(|*Obj*|).
if $a = 0 \Rightarrow \mathfrak{F}$(|*ObjMap*|)(|0|)
| $a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F}$(|*ObjMap*|)(|0|)
| $a = 2_{\mathbb{N}} \Rightarrow \mathfrak{F}$(|*ObjMap*|)(|$1_{\mathbb{N}}$|)
| *otherwise* $\Rightarrow \mathfrak{F}$(|*HomCod*|)(|*Obj*|)
)
and *LK23* $\mathfrak{F}$(|*ArrMap*|) =
(
$\lambda f \in_{\circ}$ *cat-ordinal* ($3_{\mathbb{N}}$)(|*Arr*|).
if $f = [0, 0]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, 0|)$_{\bullet}$
| $f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, 0|)$_{\bullet}$
| $f = [0, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}$(|*ArrMap*|)(|0, $1_{\mathbb{N}}$|)$_{\bullet}$

```
        | f = [1_ℕ, 1_ℕ]_∘ ⇒ 𝔉(|ArrMap|)(|0, 0|)•
        | f = [1_ℕ, 2_ℕ]_∘ ⇒ 𝔉(|ArrMap|)(|0, 1_ℕ|)•
        | f = [2_ℕ, 2_ℕ]_∘ ⇒ 𝔉(|ArrMap|)(|1_ℕ, 1_ℕ|)•
        | otherwise ⇒ 𝔉(|HomCod|)(|Arr|)
    )
  and LK23 𝔉(|HomDom|) = cat-ordinal (3_ℕ)
  and LK23 𝔉(|HomCod|) = 𝔉(|HomCod|)
 unfolding LK23-def dghm-field-simps by (simp-all add: nat-omega-simps)

context is-functor
begin

lemmas LK23-components′ = LK23-components[where 𝔉=𝔉, unfolded cat-cs-simps]

lemmas [cat-Kan-cs-simps] = LK23-components′(3,4)

end

lemmas [cat-Kan-cs-simps] = is-functor.LK23-components′(3,4)
```

16.3.2 Object map

```
mk-VLambda LK23-components(1)
 |vsv LK23-ObjMap-vsv[cat-Kan-cs-intros]|
 |vdomain LK23-ObjMap-vdomain[cat-Kan-cs-simps]|
 |app LK23-ObjMap-app|

lemma LK23-ObjMap-app-0[cat-Kan-cs-simps]:
 assumes a = 0
 shows LK23 𝔉(|ObjMap|)(|a|) = 𝔉(|ObjMap|)(|0|)
 unfolding LK23-components assms cat-ordinal-components by simp

lemma LK23-ObjMap-app-1[cat-Kan-cs-simps]:
 assumes a = 1_ℕ
 shows LK23 𝔉(|ObjMap|)(|a|) = 𝔉(|ObjMap|)(|0|)
 unfolding LK23-components assms cat-ordinal-components by simp

lemma LK23-ObjMap-app-2[cat-Kan-cs-simps]:
 assumes a = 2_ℕ
 shows LK23 𝔉(|ObjMap|)(|a|) = 𝔉(|ObjMap|)(|1_ℕ|)
 unfolding LK23-components assms cat-ordinal-components by simp
```

16.3.3 Arrow map

```
mk-VLambda LK23-components(2)
 |vsv LK23-ArrMap-vsv[cat-Kan-cs-intros]|
 |vdomain LK23-ArrMap-vdomain[cat-Kan-cs-simps]|
 |app LK23-ArrMap-app|

lemma LK23-ArrMap-app-00[cat-Kan-cs-simps]:
 assumes f = [0, 0]_∘
 shows LK23 𝔉(|ArrMap|)(|f|) = 𝔉(|ArrMap|)(|0, 0|)•
proof–
 from 0123 have f: f ∈_∘ cat-ordinal (3_ℕ)(|Arr|)
  by
   (
    cs-concl cs-shallow
     cs-intro: V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
```

```
    )
  then show ?thesis unfolding LK23-components assms by auto
qed

lemma LK23-ArrMap-app-01[cat-Kan-cs-simps]:
  assumes f = [0, 1_ℕ]_∘
  shows LK23 ℱ(|ArrMap|)(|f|) = ℱ(|ArrMap|)(|0, 0|)•
proof–
  from 0123 have f: f ∈_∘ cat-ordinal (3_ℕ)(|Arr|)
    by
      (
        cs-concl cs-shallow
          cs-intro: V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
      )
  then show ?thesis unfolding LK23-components assms by auto
qed

lemma LK23-ArrMap-app-02[cat-Kan-cs-simps]:
  assumes f = [0, 2_ℕ]_∘
  shows LK23 ℱ(|ArrMap|)(|f|) = ℱ(|ArrMap|)(|0, 1_ℕ|)•
proof–
  from 0123 have f: f ∈_∘ cat-ordinal (3_ℕ)(|Arr|)
    by
      (
        cs-concl cs-shallow
          cs-intro: V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
      )
  then show ?thesis unfolding LK23-components assms by auto
qed

lemma LK23-ArrMap-app-11[cat-Kan-cs-simps]:
  assumes f = [1_ℕ, 1_ℕ]_∘
  shows LK23 ℱ(|ArrMap|)(|f|) = ℱ(|ArrMap|)(|0, 0|)•
proof–
  from 0123 have f: f ∈_∘ cat-ordinal (3_ℕ)(|Arr|)
    by
      (
        cs-concl cs-shallow
          cs-intro: V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
      )
  then show ?thesis unfolding LK23-components assms by auto
qed

lemma LK23-ArrMap-app-12[cat-Kan-cs-simps]:
  assumes f = [1_ℕ, 2_ℕ]_∘
  shows LK23 ℱ(|ArrMap|)(|f|) = ℱ(|ArrMap|)(|0, 1_ℕ|)•
proof–
  from 0123 have f: f ∈_∘ cat-ordinal (3_ℕ)(|Arr|)
    by
      (
        cs-concl
          cs-simp: omega-of-set
          cs-intro: nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms
      )
  then show ?thesis unfolding LK23-components assms by auto
qed

lemma LK23-ArrMap-app-22[cat-Kan-cs-simps]:
```

assumes $f = [2_{\mathbb{N}},\ 2_{\mathbb{N}}]_{\circ}$
shows *LK23* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor 1_{\mathbb{N}},\ 1_{\mathbb{N}} \rfloor)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ}$ *cat-ordinal* $(3_{\mathbb{N}})(\lfloor Arr \rfloor)$
by
(
cs-concl **cs-shallow**
cs-intro: *nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms*
)
then show *?thesis* **unfolding** *LK23-components assms* **by** *simp*
qed

16.3.4 *LK23* is a functor

lemma *cat-LK23-is-functor*:
assumes $\mathfrak{F} :$ *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *LK23* $\mathfrak{F} :$ *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

interpret $\mathfrak{F}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))

from *ord-of-nat-ω* **interpret** *cat-ordinal-2*: *finite-category* α ‹*cat-ordinal* $(2_{\mathbb{N}})$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-ordinal-cs-intros*)
from *ord-of-nat-ω* **interpret** *cat-ordinal-3*: *finite-category* α ‹*cat-ordinal* $(3_{\mathbb{N}})$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-ordinal-cs-intros*)

interpret $\mathfrak{F}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*)

show *?thesis*
proof(*intro is-functorI'*)
show *vfsequence* (*LK23* $\mathfrak{F}$) **unfolding** *LK23-def* **by** *auto*
show *vcard* (*LK23* $\mathfrak{F}$) = $4_{\mathbb{N}}$ **unfolding** *LK23-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*LK23* $\mathfrak{F}(\lfloor ObjMap \rfloor)$) $\subseteq_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-Kan-cs-simps*)
fix *x* **assume** *prems*: $x \in_{\circ}$ *cat-ordinal* $(3_{\mathbb{N}})(\lfloor Obj \rfloor)$
then consider ‹$x = 0$› | ‹$x = 1_{\mathbb{N}}$› | ‹$x = 2_{\mathbb{N}}$›
unfolding *cat-ordinal-cs-simps three* **by** *auto*
then show *LK23* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
by *cases*
(
cs-concl
cs-simp: *cat-Kan-cs-simps cat-ordinal-cs-simps omega-of-set*
cs-intro: *cat-cs-intros nat-omega-intros*
)+
qed (*cs-concl* **cs-shallow cs-intro**: *cat-Kan-cs-intros*)
show *LK23* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) :$ *LK23* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{C}}$ *LK23* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
if $f : a \mapsto_{cat\text{-}ordinal\ (3_{\mathbb{N}})} b$ **for** *a b f*
proof–
from *0123 that* **show** *?thesis*
by (*elim cat-ordinal-3-is-arrE*; *simp only*:)
(
cs-concl
cs-simp: *cat-Kan-cs-simps*
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)+
qed
show
LK23 $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \circ_{A cat\text{-}ordinal\ (3_{\mathbb{N}})} f \rfloor) =$

```
      LK23 𝔉(|ArrMap|)(|g|) ∘_A ℭ LK23 𝔉(|ArrMap|)(|f|)
    if g : b ↦_cat-ordinal (3_ℕ) c and f : a ↦_cat-ordinal (3_ℕ) b
    for b c g a f
  proof–
    from 0123 that show ?thesis
      by (elim cat-ordinal-3-is-arrE; simp only:; (solves‹simp›)?)
        (
          cs-concl
            cs-simp:
              cat-ordinal-cs-simps
              cat-Kan-cs-simps
              𝔉.cf-ArrMap-Comp[symmetric]
            cs-intro: V-cs-intros cat-cs-intros cat-ordinal-cs-intros
        )+
  qed
  show LK23 𝔉(|ArrMap|)(|cat-ordinal (3_ℕ)(|CId|)(|c|)|) = ℭ(|CId|)(|LK23 𝔉(|ObjMap|)(|c|)|)
    if c ∈_∘ cat-ordinal (3_ℕ)(|Obj|) for c
  proof–
    from that consider ‹c = 0› | ‹c = 1_ℕ› | ‹c = 2_ℕ›
      unfolding cat-ordinal-components three by auto
    moreover have 0 ∈_∘ 2_ℕ 1_ℕ ∈_∘ 2_ℕ 0 ∈_∘ 3_ℕ 1_ℕ ∈_∘ 3_ℕ 2_ℕ ∈_∘ 3_ℕ by auto
    ultimately show ?thesis
      by (cases, use nothing in ‹simp-all only:›)
        (
          cs-concl
            cs-simp:
              cat-ordinal-cs-simps
              cat-Kan-cs-simps
              is-functor.cf-ObjMap-CId[symmetric]
            cs-intro: V-cs-intros cat-cs-intros cat-ordinal-cs-intros
        )+
  qed
qed
  (
    cs-concl cs-shallow
      cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
  )+

qed
```

```
lemma cat-LK23-is-functor'[cat-Kan-cs-intros]:
  assumes 𝔉 : cat-ordinal (2_ℕ) ↦↦_Cα ℭ
    and 𝔄' = cat-ordinal (3_ℕ)
  shows LK23 𝔉 : 𝔄' ↦↦_Cα ℭ
  using assms(1) unfolding assms(2) by (rule cat-LK23-is-functor)
```

16.3.5 The fundamental property of *LK23*

```
lemma cf-comp-LK23-ℜ23[cat-Kan-cs-simps]:
  assumes 𝔉 : cat-ordinal (2_ℕ) ↦↦_Cα ℭ
  shows LK23 𝔉 ∘_CF ℜ23 = 𝔉
proof–

  interpret 𝔉: is-functor α ‹cat-ordinal (2_ℕ)› ℭ 𝔉 by (rule assms(1))
  interpret ℜ23: is-functor α ‹cat-ordinal (2_ℕ)› ‹cat-ordinal (3_ℕ)› ‹ℜ23›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros)
  interpret LK23: is-functor α ‹cat-ordinal (3_ℕ)› ℭ ‹LK23 𝔉›
    by (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)
```

```
  show ?thesis
  proof(rule cf-eqI)
    show 𝔉 : cat-ordinal (2_ℕ) ↦↦_Cα ℭ by (rule assms)
    have ObjMap-dom-lhs: 𝒟∘ ((LK23 𝔉 ∘_CF ℜ23)(⦇ObjMap⦈)) = 2_ℕ
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-ordinal-cs-simps cs-intro: cat-cs-intros
        )
    have ObjMap-dom-rhs: 𝒟∘ (𝔉(⦇ObjMap⦈)) = 2_ℕ
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
    show (LK23 𝔉 ∘_CF ℜ23)(⦇ObjMap⦈) = 𝔉(⦇ObjMap⦈)
    proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
      fix a assume prems: a ∈∘ 2_ℕ
      then consider ‹a = 0› | ‹a = 1_ℕ› by force
      then show (LK23 𝔉 ∘_CF ℜ23)(⦇ObjMap⦈)(⦇a⦈) = 𝔉(⦇ObjMap⦈)(⦇a⦈)
        by (cases, use nothing in ‹simp-all only:›)
          (
            cs-concl
              cs-simp:
                omega-of-set cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps
              cs-intro: cat-cs-intros nat-omega-intros
          )+
    qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)+
    have ArrMap-dom-lhs: 𝒟∘ ((LK23 𝔉 ∘_CF ℜ23)(⦇ArrMap⦈)) = cat-ordinal (2_ℕ)(⦇Arr⦈)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    have ArrMap-dom-rhs: 𝒟∘ (𝔉(⦇ArrMap⦈)) = cat-ordinal (2_ℕ)(⦇Arr⦈)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show (LK23 𝔉 ∘_CF ℜ23)(⦇ArrMap⦈) = 𝔉(⦇ArrMap⦈)
    proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
      fix f assume prems: f ∈∘ cat-ordinal (2_ℕ)(⦇Arr⦈)
      then obtain a b where f : a ↦_cat-ordinal (2_ℕ) b by auto
      then show (LK23 𝔉 ∘_CF ℜ23)(⦇ArrMap⦈)(⦇f⦈) = 𝔉(⦇ArrMap⦈)(⦇f⦈)
        by (elim cat-ordinal-2-is-arrE; simp only:)
          (
            cs-concl
              cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
          )+
    qed (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros)+
  qed (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)

qed
```

16.4 *RK23*: the functor associated with the right Kan extension along $\mathfrak{K}23$

16.4.1 Definition and elementary properties

```
definition RK23 :: V ⇒ V
  where RK23 𝔉 =
    [
      (
        λa∈∘cat-ordinal (3_ℕ)(⦇Obj⦈).
          if a = 0 ⇒ 𝔉(⦇ObjMap⦈)(⦇0⦈)
          | a = 1_ℕ ⇒ 𝔉(⦇ObjMap⦈)(⦇1_ℕ⦈)
          | a = 2_ℕ ⇒ 𝔉(⦇ObjMap⦈)(⦇1_ℕ⦈)
          | otherwise ⇒ 𝔉(⦇HomCod⦈)(⦇Obj⦈)
      ),
```

```
    (
      λf∈∘cat-ordinal (3ℕ)(|Arr|).
       if f = [0, 0]∘ ⇒ 𝔉(|ArrMap|)(|0, 0|)•
        | f = [0, 1ℕ]∘ ⇒ 𝔉(|ArrMap|)(|0, 1ℕ|)•
        | f = [0, 2ℕ]∘ ⇒ 𝔉(|ArrMap|)(|0, 1ℕ|)•
        | f = [1ℕ, 1ℕ]∘ ⇒ 𝔉(|ArrMap|)(|1ℕ, 1ℕ|)•
        | f = [1ℕ, 2ℕ]∘ ⇒ 𝔉(|ArrMap|)(|1ℕ, 1ℕ|)•
        | f = [2ℕ, 2ℕ]∘ ⇒ 𝔉(|ArrMap|)(|1ℕ, 1ℕ|)•
        | otherwise ⇒ 𝔉(|HomCod|)(|Arr|)
    ),
    cat-ordinal (3ℕ),
    𝔉(|HomCod|)
  ]∘
```

Components.

```
lemma RK23-components:
  shows RK23 𝔉(|ObjMap|) =
    (
      λa∈∘cat-ordinal (3ℕ)(|Obj|).
        if a = 0 ⇒ 𝔉(|ObjMap|)(|0|)
         | a = 1ℕ ⇒ 𝔉(|ObjMap|)(|1ℕ|)
         | a = 2ℕ ⇒ 𝔉(|ObjMap|)(|1ℕ|)
         | otherwise ⇒ 𝔉(|HomCod|)(|Obj|)
    )
    and RK23 𝔉(|ArrMap|) =
      (
        λf∈∘cat-ordinal (3ℕ)(|Arr|).
         if f = [0, 0]∘ ⇒ 𝔉(|ArrMap|)(|0, 0|)•
          | f = [0, 1ℕ]∘ ⇒ 𝔉(|ArrMap|)(|0, 1ℕ|)•
          | f = [0, 2ℕ]∘ ⇒ 𝔉(|ArrMap|)(|0, 1ℕ|)•
          | f = [1ℕ, 1ℕ]∘ ⇒ 𝔉(|ArrMap|)(|1ℕ, 1ℕ|)•
          | f = [1ℕ, 2ℕ]∘ ⇒ 𝔉(|ArrMap|)(|1ℕ, 1ℕ|)•
          | f = [2ℕ, 2ℕ]∘ ⇒ 𝔉(|ArrMap|)(|1ℕ, 1ℕ|)•
          | otherwise ⇒ 𝔉(|HomCod|)(|Arr|)
      )
    and RK23 𝔉(|HomDom|) = cat-ordinal (3ℕ)
    and RK23 𝔉(|HomCod|) = 𝔉(|HomCod|)
  unfolding RK23-def dghm-field-simps by (simp-all add: nat-omega-simps)

context is-functor
begin

lemmas RK23-components′ = RK23-components[where 𝔉=𝔉, unfolded cat-cs-simps]

lemmas [cat-Kan-cs-simps] = RK23-components′(3,4)

end

lemmas [cat-Kan-cs-simps] = is-functor.RK23-components′(3,4)
```

16.4.2 Object map

```
mk-VLambda RK23-components(1)
  |vsv RK23-ObjMap-vsv[cat-Kan-cs-intros]|
  |vdomain RK23-ObjMap-vdomain[cat-Kan-cs-simps]|
  |app RK23-ObjMap-app|

lemma RK23-ObjMap-app-0[cat-Kan-cs-simps]:
```

assumes $a = 0$
shows $RK23\ \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|0|\!)$
unfolding *RK23-components assms cat-ordinal-components* **by** *simp*

lemma *RK23-ObjMap-app-1*[*cat-Kan-cs-simps*]:
assumes $a = 1_{\mathbb{N}}$
shows $RK23\ \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|1_{\mathbb{N}}|\!)$
unfolding *RK23-components assms cat-ordinal-components* **by** *simp*

lemma *RK23-ObjMap-app-2*[*cat-Kan-cs-simps*]:
assumes $a = 2_{\mathbb{N}}$
shows $RK23\ \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|1_{\mathbb{N}}|\!)$
unfolding *RK23-components assms cat-ordinal-components* **by** *simp*

16.4.3 Arrow map

mk-VLambda *RK23-components*(*2*)
|*vsv RK23-ArrMap-vsv*[*cat-Kan-cs-intros*]|
|*vdomain RK23-ArrMap-vdomain*[*cat-Kan-cs-simps*]|
|*app RK23-ArrMap-app*|

lemma *RK23-ArrMap-app-00*[*cat-Kan-cs-simps*]:
assumes $f = [0, 0]_{\circ}$
shows $RK23\ \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|0, 0|\!)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ} cat\text{-}ordinal\ (3_{\mathbb{N}})(\!|Arr|\!)$
by
(
cs-concl **cs-shallow cs-intro**:
V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
)
then show *?thesis* **unfolding** *RK23-components assms* **by** *auto*
qed

lemma *RK23-ArrMap-app-01*[*cat-Kan-cs-simps*]:
assumes $f = [0, 1_{\mathbb{N}}]_{\circ}$
shows $RK23\ \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|0, 1_{\mathbb{N}}|\!)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ} cat\text{-}ordinal\ (3_{\mathbb{N}})(\!|Arr|\!)$
by
(
cs-concl **cs-shallow cs-intro**:
V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
)
then show *?thesis* **unfolding** *RK23-components assms* **by** *auto*
qed

lemma *RK23-ArrMap-app-02*[*cat-Kan-cs-simps*]:
assumes $f = [0, 2_{\mathbb{N}}]_{\circ}$
shows $RK23\ \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|0, 1_{\mathbb{N}}|\!)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ} cat\text{-}ordinal\ (3_{\mathbb{N}})(\!|Arr|\!)$
by
(
cs-concl **cs-shallow cs-intro**:
V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
)
then show *?thesis* **unfolding** *RK23-components assms* **by** *auto*

qed

lemma *RK23-ArrMap-app-11*[*cat-Kan-cs-simps*]:
assumes $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$
shows *RK23* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor 1_{\mathbb{N}}, 1_{\mathbb{N}} \rfloor)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ}$ *cat-ordinal* $(3_{\mathbb{N}})(\lfloor Arr \rfloor)$
by
(
cs-concl **cs-shallow cs-intro**:
V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms
)
then show *?thesis* **unfolding** *RK23-components assms* **by** *auto*
qed

lemma *RK23-ArrMap-app-12*[*cat-Kan-cs-simps*]:
assumes $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
shows *RK23* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor 1_{\mathbb{N}}, 1_{\mathbb{N}} \rfloor)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ}$ *cat-ordinal* $(3_{\mathbb{N}})(\lfloor Arr \rfloor)$
by
(
cs-concl
cs-simp: *omega-of-set*
cs-intro: *nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms*
)
then show *?thesis* **unfolding** *RK23-components assms* **by** *auto*
qed

lemma *RK23-ArrMap-app-22*[*cat-Kan-cs-simps*]:
assumes $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
shows *RK23* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor 1_{\mathbb{N}}, 1_{\mathbb{N}} \rfloor)_{\bullet}$
proof–
from *0123* **have** *f*: $f \in_{\circ}$ *cat-ordinal* $(3_{\mathbb{N}})(\lfloor Arr \rfloor)$
by
(
cs-concl **cs-shallow cs-intro**:
nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms
)
then show *?thesis* **unfolding** *RK23-components assms* **by** *simp*
qed

16.4.4 *RK23* is a functor

lemma *cat-RK23-is-functor*:
assumes $\mathfrak{F} : \textit{cat-ordinal}\ (2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *RK23* $\mathfrak{F} : \textit{cat-ordinal}\ (3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

interpret $\mathfrak{F}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))

from *ord-of-nat-ω* **interpret** *cat-ordinal-2*: *finite-category* α ‹*cat-ordinal* $(2_{\mathbb{N}})$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-ordinal-cs-intros*)
from *ord-of-nat-ω* **interpret** *cat-ordinal-3*: *finite-category* α ‹*cat-ordinal* $(3_{\mathbb{N}})$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-ordinal-cs-intros*)

interpret $\mathfrak{F}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*)

```
show ?thesis
proof(intro is-functorI')
  show vfsequence (RK23 𝔉) unfolding RK23-def by auto
  show vcard (RK23 𝔉) = ℵ₄ unfolding RK23-def by (simp add: nat-omega-simps)
  show ℛ∘ (RK23 𝔉(|ObjMap|)) ⊆∘ ℭ(|Obj|)
  proof(rule vsv.vsv-vrange-vsubset, unfold cat-Kan-cs-simps)
    fix x assume prems: x ∈∘ cat-ordinal (3₊)(|Obj|)
    then consider ‹x = 0› | ‹x = 1₊› | ‹x = 2₊›
      unfolding cat-ordinal-cs-simps three by auto
    then show RK23 𝔉(|ObjMap|)(|x|) ∈∘ ℭ(|Obj|)
      by cases
        (
          cs-concl
            cs-simp: cat-Kan-cs-simps cat-ordinal-cs-simps omega-of-set
            cs-intro: cat-cs-intros nat-omega-intros
        )+
  qed (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)
  show RK23 𝔉(|ArrMap|)(|f|) : RK23 𝔉(|ObjMap|)(|a|) ↦ℭ RK23 𝔉(|ObjMap|)(|b|)
    if f : a ↦cat-ordinal (3₊) b for a b f
  proof–
    from 0123 that show ?thesis
      by (elim cat-ordinal-3-is-arrE; simp only:)
        (
          cs-concl
            cs-simp: cat-Kan-cs-simps
            cs-intro: V-cs-intros cat-cs-intros cat-ordinal-cs-intros
        )+
  qed
  show
    RK23 𝔉(|ArrMap|)(|g ∘A cat-ordinal (3₊) f|) =
      RK23 𝔉(|ArrMap|)(|g|) ∘Aℭ RK23 𝔉(|ArrMap|)(|f|)
    if g : b ↦cat-ordinal (3₊) c and f : a ↦cat-ordinal (3₊) b
    for b c g a f
    using 0123 that
    by (elim cat-ordinal-3-is-arrE; simp only:; (solves‹simp›)?)
      (
        cs-concl
          cs-simp:
            cat-ordinal-cs-simps
            cat-Kan-cs-simps
            𝔉.cf-ArrMap-Comp[symmetric]
          cs-intro: V-cs-intros cat-cs-intros cat-ordinal-cs-intros
      )+
  show RK23 𝔉(|ArrMap|)(|cat-ordinal (3₊)(|CId|)(|c|)|) = ℭ(|CId|)(|RK23 𝔉(|ObjMap|)(|c|)|)
    if c ∈∘ cat-ordinal (3₊)(|Obj|) for c
  proof–
    from that consider ‹c = 0› | ‹c = 1₊› | ‹c = 2₊›
      unfolding cat-ordinal-components three by auto
    then show ?thesis
      by (cases, use 0123 in ‹simp-all only:›)
        (
          cs-concl
            cs-simp:
              cat-ordinal-cs-simps
              cat-Kan-cs-simps
              is-functor.cf-ObjMap-CId[symmetric]
            cs-intro: V-cs-intros cat-cs-intros cat-ordinal-cs-intros
        )+
```

```
    qed
  qed
    (
      cs-concl cs-shallow
        cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
    )+

qed

lemma cat-RK23-is-functor'[cat-Kan-cs-intros]:
  assumes 𝔉 : cat-ordinal (2_ℕ) ↦↦_{Cα} ℭ
    and 𝔄' = cat-ordinal (3_ℕ)
  shows RK23 𝔉 : 𝔄' ↦↦_{Cα} ℭ
  using assms(1) unfolding assms(2) by (rule cat-RK23-is-functor)
```

16.4.5 The fundamental property of *RK23*

```
lemma cf-comp-RK23-ℜ23[cat-Kan-cs-simps]:
  assumes 𝔉 : cat-ordinal (2_ℕ) ↦↦_{Cα} ℭ
  shows RK23 𝔉 ∘_{CF} ℜ23 = 𝔉
proof–

  interpret 𝔉: is-functor α ‹cat-ordinal (2_ℕ)› ℭ 𝔉 by (rule assms(1))
  interpret ℜ23: is-functor α ‹cat-ordinal (2_ℕ)› ‹cat-ordinal (3_ℕ)› ‹ℜ23›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros)
  interpret RK23: is-functor α ‹cat-ordinal (3_ℕ)› ℭ ‹RK23 𝔉›
    by (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)

  show ?thesis
  proof(rule cf-eqI)
    show 𝔉 : cat-ordinal (2_ℕ) ↦↦_{Cα} ℭ by (rule assms)
    have ObjMap-dom-lhs: 𝒟_∘ ((RK23 𝔉 ∘_{CF} ℜ23)(|ObjMap|)) = 2_ℕ
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-ordinal-cs-simps cs-intro: cat-cs-intros
        )
    have ObjMap-dom-rhs: 𝒟_∘ (𝔉(|ObjMap|)) = 2_ℕ
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
    show (RK23 𝔉 ∘_{CF} ℜ23)(|ObjMap|) = 𝔉(|ObjMap|)
    proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
      fix a assume prems: a ∈_∘ 2_ℕ
      then consider ‹a = 0› | ‹a = 1_ℕ› by force
      then show (RK23 𝔉 ∘_{CF} ℜ23)(|ObjMap|)(|a|) = 𝔉(|ObjMap|)(|a|)
        by (cases, use nothing in ‹simp-all only:›)
          (
            cs-concl
              cs-simp:
                omega-of-set cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps
              cs-intro: cat-cs-intros nat-omega-intros
          )+
    qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)+
    have ArrMap-dom-lhs: 𝒟_∘ ((RK23 𝔉 ∘_{CF} ℜ23)(|ArrMap|)) = cat-ordinal (2_ℕ)(|Arr|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    have ArrMap-dom-rhs: 𝒟_∘ (𝔉(|ArrMap|)) = cat-ordinal (2_ℕ)(|Arr|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show (RK23 𝔉 ∘_{CF} ℜ23)(|ArrMap|) = 𝔉(|ArrMap|)
    proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
```

fix f **assume** *prems*: $f \in_\circ$ *cat-ordinal* $(2_{\mathbb{N}})(\!|Arr|\!)$
then obtain a b **where** $f : a \mapsto_{\textit{cat-ordinal } (2_{\mathbb{N}})} b$ **by** *auto*
then show $(\mathit{RK23}\ \mathfrak{F} \circ_{CF} \mathfrak{K}23)(\!|\mathit{ArrMap}|\!)(\!|f|\!) = \mathfrak{F}(\!|\mathit{ArrMap}|\!)(\!|f|\!)$
by (*elim cat-ordinal-2-is-arrE*; *simp only*:)
(
cs-concl
cs-simp: *cat-cs-simps cat-Kan-cs-simps* **cs-intro**: *cat-cs-intros*
)+
qed (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros V-cs-intros*)+
qed (*cs-concl* **cs-intro**: *cat-Kan-cs-intros cat-cs-intros*)

qed

16.5 *RK-σ23*: towards the universal property of the right Kan extension along *𝔎23*

16.5.1 Definition and elementary properties

definition $\mathit{RK\text{-}\sigma 23} :: V \Rightarrow V \Rightarrow V \Rightarrow V$
where $\mathit{RK\text{-}\sigma 23}\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}' =$
[
(
$\lambda a \in_\circ \textit{cat-ordinal } (3_{\mathbb{N}})(\!|\mathit{Obj}|\!).$
if $a = 0 \Rightarrow \varepsilon'(\!|\mathit{NTMap}|\!)(\!|0|\!)$
$\mid a = 1_{\mathbb{N}} \Rightarrow \varepsilon'(\!|\mathit{NTMap}|\!)(\!|1_{\mathbb{N}}|\!}) \circ_{A\mathfrak{T}(\!|\mathit{HomCod}|\!)} \mathfrak{F}'(\!|\mathit{ArrMap}|\!)(\!|1_{\mathbb{N}}, 2_{\mathbb{N}}|\!)_\bullet$
$\mid a = 2_{\mathbb{N}} \Rightarrow \varepsilon'(\!|\mathit{NTMap}|\!)(\!|1_{\mathbb{N}}|\!)$
$\mid \mathit{otherwise} \Rightarrow \mathfrak{T}(\!|\mathit{HomCod}|\!)(\!|\mathit{Arr}|\!)$
),
$\mathfrak{F}'$,
RK23 $\mathfrak{T}$,
cat-ordinal $(3_{\mathbb{N}})$,
$\mathfrak{F}'(\!|\mathit{HomCod}|\!)$
]$_\circ$

Components.

lemma *RK-σ23-components*:
shows $\mathit{RK\text{-}\sigma 23}\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\!|\mathit{NTMap}|\!) =$
(
$\lambda a \in_\circ \textit{cat-ordinal } (3_{\mathbb{N}})(\!|\mathit{Obj}|\!).$
if $a = 0 \Rightarrow \varepsilon'(\!|\mathit{NTMap}|\!)(\!|0|\!)$
$\mid a = 1_{\mathbb{N}} \Rightarrow \varepsilon'(\!|\mathit{NTMap}|\!)(\!|1_{\mathbb{N}}|\!}) \circ_{A\mathfrak{T}(\!|\mathit{HomCod}|\!)} \mathfrak{F}'(\!|\mathit{ArrMap}|\!)(\!|1_{\mathbb{N}}, 2_{\mathbb{N}}|\!)_\bullet$
$\mid a = 2_{\mathbb{N}} \Rightarrow \varepsilon'(\!|\mathit{NTMap}|\!)(\!|1_{\mathbb{N}}|\!)$
$\mid \mathit{otherwise} \Rightarrow \mathfrak{T}(\!|\mathit{HomCod}|\!)(\!|\mathit{Arr}|\!)$
)
and $\mathit{RK\text{-}\sigma 23}\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\!|\mathit{NTDom}|\!) = \mathfrak{F}'$
and $\mathit{RK\text{-}\sigma 23}\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\!|\mathit{NTCod}|\!) = \mathit{RK23}\ \mathfrak{T}$
and $\mathit{RK\text{-}\sigma 23}\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\!|\mathit{NTDGDom}|\!) = \textit{cat-ordinal } (3_{\mathbb{N}})$
and $\mathit{RK\text{-}\sigma 23}\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\!|\mathit{NTDGCod}|\!) = \mathfrak{F}'(\!|\mathit{HomCod}|\!)$
unfolding *RK-σ23-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{A}$ $\mathfrak{F}'$ $\mathfrak{T}$
assumes $\mathfrak{F}'$: $\mathfrak{F}' : \textit{cat-ordinal } (3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{T}$: $\mathfrak{T} : \textit{cat-ordinal } (2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{F}'$: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{F}'$ **by** (*rule* $\mathfrak{F}'$)
interpretation $\mathfrak{T}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *RK-σ23-components′* =
RK-σ23-components[**where** $\mathfrak{F}'$=$\mathfrak{F}'$ **and** $\mathfrak{T}$=$\mathfrak{T}$, *unfolded cat-cs-simps*]

lemmas [*cat-Kan-cs-simps*] = *RK-σ23-components′*(*2–5*)

end

16.5.2 Natural transformation map

mk-VLambda *RK-σ23-components*(*1*)
|*vsv RK-σ23-NTMap-vsv*[*cat-Kan-cs-intros*]|
|*vdomain RK-σ23-NTMap-vdomain*[*cat-Kan-cs-simps*]|
|*app RK-σ23-NTMap-app*|

lemma *RK-σ23-NTMap-app-0*[*cat-Kan-cs-simps*]:
assumes *a* = *0*
shows *RK-σ23* $\mathfrak{T}$ ε' $\mathfrak{F}'(\!|NTMap|\!)(\!|a|\!)$ = $\varepsilon'(\!|NTMap|\!)(\!|0|\!)$
using *assms* **unfolding** *RK-σ23-components cat-ordinal-cs-simps* **by** *simp*

lemma (**in** *is-functor*) *RK-σ23-NTMap-app-1*[*cat-Kan-cs-simps*]:
assumes *a* = $1_{\mathbb{N}}$
shows *RK-σ23* $\mathfrak{F}$ ε' $\mathfrak{F}'(\!|NTMap|\!)(\!|a|\!)$ = $\varepsilon'(\!|NTMap|\!)(\!|1_{\mathbb{N}}|\!)$ $\circ_{A\mathfrak{B}}$ $\mathfrak{F}'(\!|ArrMap|\!)(\!|1_{\mathbb{N}}, 2_{\mathbb{N}}|\!)_{\bullet}$
using *assms*
unfolding *RK-σ23-components cat-ordinal-cs-simps cat-cs-simps*
by *simp*

lemmas [*cat-Kan-cs-simps*] = *is-functor.RK-σ23-NTMap-app-1*

lemma *RK-σ23-NTMap-app-2*[*cat-Kan-cs-simps*]:
assumes *a* = $2_{\mathbb{N}}$
shows *RK-σ23* $\mathfrak{T}$ ε' $\mathfrak{F}'(\!|NTMap|\!)(\!|a|\!)$ = $\varepsilon'(\!|NTMap|\!)(\!|1_{\mathbb{N}}|\!)$
using *assms* **unfolding** *RK-σ23-components cat-ordinal-cs-simps* **by** *simp*

16.5.3 *RK-σ23* is a natural transformation

lemma *RK-σ23-is-ntcf*:
assumes $\mathfrak{F}'$: *cat-ordinal* $(3_{\mathbb{N}})$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
and $\mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}})$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
and ε' : $\mathfrak{F}'$ $\circ_{CF}$ $\mathfrak{K}23$ $\mapsto_{CF}$ $\mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}})$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
shows *RK-σ23* $\mathfrak{T}$ ε' $\mathfrak{F}'$: $\mathfrak{F}'$ $\mapsto_{CF}$ *RK23* $\mathfrak{T}$: *cat-ordinal* $(3_{\mathbb{N}})$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
proof–

interpret $\mathfrak{F}'$: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{F}'$ **by** (*rule assms*(*1*))
interpret $\mathfrak{T}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*2*))
interpret ε': *is-ntcf* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ ‹$\mathfrak{F}'$ $\circ_{CF}$ $\mathfrak{K}23$› $\mathfrak{T}$ ε'
by (*rule assms*(*3*))

interpret $\mathfrak{K}23$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› ‹*cat-ordinal* $(3_{\mathbb{N}})$› ‹$\mathfrak{K}23$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Kan-cs-intros*)
interpret *RK23*: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ ‹*RK23* $\mathfrak{T}$›
by (*cs-concl* **cs-intro**: *cat-Kan-cs-intros cat-cs-intros*)

from *0123* **have** [*cat-cs-simps*]: $\mathfrak{T}(\!|ArrMap|\!)(\!|1_{\mathbb{N}}, 1_{\mathbb{N}}|\!)_{\bullet}$ = $\mathfrak{A}(\!|CId|\!)(\mathfrak{T}(\!|ObjMap|\!)(\!|1_{\mathbb{N}}|\!))$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-ordinal-cs-simps is-functor.cf-ObjMap-CId*[*symmetric*]

```
      cs-intro: cat-cs-intros
    )

  show ?thesis
  proof(rule is-ntcfI')
    show vfsequence (RK-σ23 𝔗 ε' 𝔉') unfolding RK-σ23-def by simp
    show vcard (RK-σ23 𝔗 ε' 𝔉') = 5ℕ
      unfolding RK-σ23-def by (simp-all add: nat-omega-simps)
    show RK-σ23 𝔗 ε' 𝔉'(|NTMap|)(|a|) : 𝔉'(|ObjMap|)(|a|) ↦𝔄 RK23 𝔗(|ObjMap|)(|a|)
      if a ∈∘ cat-ordinal (3ℕ)(|Obj|) for a
    proof–
      from that consider ‹a = 0› | ‹a = 1ℕ› | ‹a = 2ℕ›
        unfolding cat-ordinal-cs-simps three by auto
      from this 0123 show ?thesis
        by (cases, use nothing in ‹simp-all only:›)
          (
            cs-concl
              cs-simp: cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps
              cs-intro:
                cat-cs-intros
                cat-ordinal-cs-intros
                cat-Kan-cs-intros
                nat-omega-intros
          )+
    qed
    show
      RK-σ23 𝔗 ε' 𝔉'(|NTMap|)(|b|) ∘A𝔄 𝔉'(|ArrMap|)(|f|) =
        RK23 𝔗(|ArrMap|)(|f|) ∘A𝔄 RK-σ23 𝔗 ε' 𝔉'(|NTMap|)(|a|)
      if f : a ↦cat-ordinal (3ℕ) b for a b f
      using that 0123
      by (elim cat-ordinal-3-is-arrE, use nothing in ‹simp-all only:›)
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-ordinal-cs-simps
              𝔉'.cf-ArrMap-Comp[symmetric]
              𝔉'.HomCod.cat-Comp-assoc
              ε'.ntcf-Comp-commute[symmetric]
              cat-Kan-cs-simps
            cs-intro: cat-cs-intros cat-ordinal-cs-intros nat-omega-intros
        )+
  qed
    (
      cs-concl
        cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
    )+

qed

lemma RK-σ23-is-ntcf'[cat-Kan-cs-intros]:
  assumes 𝔉' : cat-ordinal (3ℕ) ↦↦Cα 𝔄
    and 𝔗 : cat-ordinal (2ℕ) ↦↦Cα 𝔄
    and ε' : 𝔉' ∘CF 𝔎23 ↦CF 𝔗 : cat-ordinal (2ℕ) ↦↦Cα 𝔄
    and 𝔊' = 𝔉'
    and ℌ' = RK23 𝔗
    and ℭ' = cat-ordinal (3ℕ)
  shows RK-σ23 𝔗 ε' 𝔉' : 𝔊' ↦CF ℌ' : ℭ' ↦↦Cα 𝔄
```

using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule RK-σ23-is-ntcf*)

16.6 The right Kan extension along $\mathfrak{K}23$

lemma *ε23-is-cat-rKe*:
assumes $\mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
shows *ntcf-id* $\mathfrak{T}$:
$RK23\ \mathfrak{T} \circ_{CF} \mathfrak{K}23 \mapsto_{CF.rKe\alpha} \mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto_{C}$ *cat-ordinal* $(3_{\mathbb{N}}) \mapsto_{C} \mathfrak{A}$
proof–

interpret $\mathfrak{T}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{K}23$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› ‹*cat-ordinal* $(3_{\mathbb{N}})$› ‹$\mathfrak{K}23$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Kan-cs-intros*)
interpret *RK23*: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ ‹*RK23* $\mathfrak{T}$›
by (*cs-concl* **cs-intro**: *cat-Kan-cs-intros cat-cs-intros*)

from *0123* **have** [*cat-cs-simps*]: $\mathfrak{T}(\!|ArrMap|\!)(\!|1_{\mathbb{N}},\ 1_{\mathbb{N}}|\!)_{\bullet} = \mathfrak{A}(\!|CId|\!)(\!|\mathfrak{T}(\!|ObjMap|\!)(\!|1_{\mathbb{N}}|\!)|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-ordinal-cs-simps is-functor.cf-ObjMap-CId*[*symmetric*]
cs-intro: *cat-cs-intros*
)

show *?thesis*
proof(*intro is-cat-rKeI′*)

fix $\mathfrak{F}'$ ε' **assume** *prems*:
$\mathfrak{F}'$: *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
ε' : $\mathfrak{F}' \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$

interpret $\mathfrak{F}'$: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{F}'$ **by** (*rule prems*(*1*))
interpret ε': *is-ntcf* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ ‹$\mathfrak{F}' \circ_{CF} \mathfrak{K}23$› $\mathfrak{T}$ ε'
by (*rule prems*(*2*))
interpret *RK-σ23*: *is-ntcf* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{F}'$ ‹*RK23* $\mathfrak{T}$› ‹*RK-σ23* $\mathfrak{T}$ ε' $\mathfrak{F}'$›
by (*intro RK-σ23-is-ntcf prems assms*)

show $\exists!\sigma$.
σ : $\mathfrak{F}' \mapsto_{CF}$ *RK23* $\mathfrak{T}$: *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$ ∧
$\varepsilon' = ntcf\text{-}id\ \mathfrak{T} \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K}23)$
proof(*intro ex1I conjI*; (*elim conjE*)*?*)
show *RK-σ23* $\mathfrak{T}$ ε' $\mathfrak{F}'$: $\mathfrak{F}' \mapsto_{CF}$ *RK23* $\mathfrak{T}$: *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
by (*intro RK-σ23.is-ntcf-axioms*)
show $\varepsilon' = ntcf\text{-}id\ \mathfrak{T} \cdot_{NTCF} (RK\text{-}\sigma23\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{K}23)$
proof(*rule ntcf-eqI*)
show ε' : $\mathfrak{F}' \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
by (*intro prems*)
then have *dom-lhs*: $\mathcal{D}_{\circ}$ $(\varepsilon'(\!|NTMap|\!)) = 2_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-ordinal-cs-simps cat-cs-simps*)
show *rhs*:
ntcf-id $\mathfrak{T} \cdot_{NTCF} (RK\text{-}\sigma23\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{K}23)$:
$\mathfrak{F}' \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\!\!\mapsto_{C\alpha} \mathfrak{A}$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Kan-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

```
  then have dom-rhs:
    𝒟∘ ((ntcf-id 𝔗 ⋆NTCF (RK-σ23 𝔗 ε' 𝔉' ∘NTCF-CF ℜ23))(⟦NTMap⟧)) = 2ℕ
    by (cs-concl cs-simp: cat-ordinal-cs-simps cat-cs-simps)
  show ε'(⟦NTMap⟧) = (ntcf-id 𝔗 ⋆NTCF (RK-σ23 𝔗 ε' 𝔉' ∘NTCF-CF ℜ23))(⟦NTMap⟧)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume prems: a ∈∘ 2ℕ
    then consider ‹a = 0› | ‹a = 1ℕ› unfolding two by auto
    then show
      ε'(⟦NTMap⟧)(⟦a⟧) =
        (ntcf-id 𝔗 ⋆NTCF (RK-σ23 𝔗 ε' 𝔉' ∘NTCF-CF ℜ23))(⟦NTMap⟧)(⟦a⟧)
      by (cases; use nothing in ‹simp-all only:›)
        (
          cs-concl
            cs-simp:
              omega-of-set
              cat-Kan-cs-simps
              cat-cs-simps
              cat-ordinal-cs-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros nat-omega-intros
        )+
  qed
    (
      use rhs in
        ‹cs-concl cs-shallow cs-intro: V-cs-intros cat-cs-intros›
    )+
qed simp-all

fix σ assume prems':
  σ : 𝔉' ↦CF RK23 𝔗 : cat-ordinal (3ℕ) ↦↦Cα 𝔄
  ε' = ntcf-id 𝔗 ⋆NTCF (σ ∘NTCF-CF ℜ23)

interpret σ: is-ntcf α ‹cat-ordinal (3ℕ)› 𝔄 𝔉' ‹RK23 𝔗› σ
  by (rule prems'(1))

from prems'(2) have
  ε'(⟦NTMap⟧)(⟦0⟧) = (ntcf-id 𝔗 ⋆NTCF (σ ∘NTCF-CF ℜ23))(⟦NTMap⟧)(⟦0⟧)
  by auto
then have [cat-cs-simps]: ε'(⟦NTMap⟧)(⟦0⟧) = σ(⟦NTMap⟧)(⟦0⟧)
  by
    (
      cs-prems cs-shallow
        cs-simp: cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps
        cs-intro: cat-cs-intros nat-omega-intros
    )
from prems'(2) have
  ε'(⟦NTMap⟧)(⟦1ℕ⟧) = (ntcf-id 𝔗 ⋆NTCF (σ ∘NTCF-CF ℜ23))(⟦NTMap⟧)(⟦1ℕ⟧)
  by auto
then have [cat-cs-simps]: ε'(⟦NTMap⟧)(⟦1ℕ⟧) = σ(⟦NTMap⟧)(⟦2ℕ⟧)
  by
    (
      cs-prems cs-shallow
        cs-simp:
          omega-of-set cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps
        cs-intro: cat-cs-intros nat-omega-intros
    )

show σ = RK-σ23 𝔗 ε' 𝔉'
proof(rule ntcf-eqI)
```

```
      show σ : 𝔉′ ↦_CF RK23 𝔗 : cat-ordinal (3_ℕ) ↦↦_Cα 𝔄
        by (rule prems′(1))
      then have dom-lhs: 𝒟∘ (σ⦇NTMap⦈) = 3_ℕ
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
      show RK-σ23 𝔗 ε′ 𝔉′ : 𝔉′ ↦_CF RK23 𝔗 : cat-ordinal (3_ℕ) ↦↦_Cα 𝔄
        by (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros)
      then have dom-rhs: 𝒟∘ (RK-σ23 𝔗 ε′ 𝔉′⦇NTMap⦈) = 3_ℕ
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
      from 0123 have 013: [0, 1_ℕ]∘ : 0 ↦_cat-ordinal (3_ℕ) 1_ℕ
        by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)
      from 0123 have 123: [1_ℕ, 2_ℕ]∘ : 1_ℕ ↦_cat-ordinal (3_ℕ) 2_ℕ
        by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)

      from σ.ntcf-Comp-commute[OF 123] 013 0123
      have [symmetric, cat-Kan-cs-simps]:
        σ⦇NTMap⦈⦇2_ℕ⦈ ∘_A𝔄 𝔉′⦇ArrMap⦈ ⦇1_ℕ, 2_ℕ⦈• = σ⦇NTMap⦈⦇1_ℕ⦈
        by
          (
            cs-prems
              cs-simp: cat-cs-simps cat-Kan-cs-simps RK23-ArrMap-app-12
              cs-intro: cat-cs-intros
          )
      show σ⦇NTMap⦈ = RK-σ23 𝔗 ε′ 𝔉′⦇NTMap⦈
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
        fix a assume prems: a ∈∘ 3_ℕ
        then consider ‹a = 0› | ‹a = 1_ℕ› | ‹a = 2_ℕ› unfolding three by auto
        then show σ⦇NTMap⦈⦇a⦈ = RK-σ23 𝔗 ε′ 𝔉′⦇NTMap⦈⦇a⦈
          by (cases; use nothing in ‹simp-all only:›)
            (cs-concl cs-simp: cat-cs-simps cat-Kan-cs-simps)+
      qed auto
    qed simp-all

  qed

qed (cs-concl cs-shallow cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)+

qed
```

16.7 *LK-σ23*: towards the universal property of the left Kan extension along *ℜ23*

16.7.1 Definition and elementary properties

```
definition LK-σ23 :: V ⇒ V ⇒ V ⇒ V
  where LK-σ23 𝔗 η′ 𝔉′ =
    [
      (
        λa∈∘cat-ordinal (3_ℕ)⦇Obj⦈.
          if a = 0 ⇒ η′⦇NTMap⦈⦇0⦈
          | a = 1_ℕ ⇒ 𝔉′⦇ArrMap⦈⦇0, 1_ℕ⦈• ∘_A𝔗⦇HomCod⦈ η′⦇NTMap⦈⦇0⦈
          | a = 2_ℕ ⇒ η′⦇NTMap⦈⦇1_ℕ⦈
          | otherwise ⇒ 𝔗⦇HomCod⦈⦇Arr⦈
      ),
      LK23 𝔗,
      𝔉′,
      cat-ordinal (3_ℕ),
      𝔉′⦇HomCod⦈
    ]∘
```

Components.

lemma *LK-σ23-components*:
shows *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTMap|\!)$ =
(
$\lambda a \in_{\circ} cat\text{-}ordinal\ (3_{\mathbb{N}})(\!|Obj|\!).$
if $a = 0 \Rightarrow \eta'(\!|NTMap|\!)(\!|0|\!)$
$\mid a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F}'(\!|ArrMap|\!)(\!|0,\ 1_{\mathbb{N}}|\!)_{\bullet} \circ_{A\mathfrak{T}(\!|HomCod|\!)}\ \eta'(\!|NTMap|\!)(\!|0|\!)$
$\mid a = 2_{\mathbb{N}} \Rightarrow \eta'(\!|NTMap|\!)(\!|1_{\mathbb{N}}|\!)$
$\mid$ *otherwise* $\Rightarrow \mathfrak{T}(\!|HomCod|\!)(\!|Arr|\!)$
)
and *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTDom|\!)$ = *LK23* $\mathfrak{T}$
and *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTCod|\!)$ = $\mathfrak{F}'$
and *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTDGDom|\!)$ = *cat-ordinal* $(3_{\mathbb{N}})$
and *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTDGCod|\!)$ = $\mathfrak{F}'(\!|HomCod|\!)$
unfolding *LK-σ23-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{A}$ $\mathfrak{F}'$ $\mathfrak{T}$
assumes $\mathfrak{F}'$: $\mathfrak{F}'$: *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{T}$: $\mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
begin

interpretation $\mathfrak{F}'$: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{F}'$ **by** (*rule* $\mathfrak{F}'$)
interpretation $\mathfrak{T}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule* $\mathfrak{T}$)

lemmas *LK-σ23-components′* =
LK-σ23-components[**where** $\mathfrak{F}'$=$\mathfrak{F}'$ **and** $\mathfrak{T}$=$\mathfrak{T}$, *unfolded cat-cs-simps*]

lemmas [*cat-Kan-cs-simps*] = *LK-σ23-components′*(*2–5*)

end

16.7.2 Natural transformation map

mk-VLambda *LK-σ23-components*(*1*)
|*vsv LK-σ23-NTMap-vsv*[*cat-Kan-cs-intros*]|
|*vdomain LK-σ23-NTMap-vdomain*[*cat-Kan-cs-simps*]|
|*app LK-σ23-NTMap-app*|

lemma *LK-σ23-NTMap-app-0*[*cat-Kan-cs-simps*]:
assumes *a* = *0*
shows *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTMap|\!)(\!|a|\!)$ = $\eta'(\!|NTMap|\!)(\!|0|\!)$
using *assms* **unfolding** *LK-σ23-components cat-ordinal-cs-simps* **by** *simp*

lemma (**in** *is-functor*) *LK-σ23-NTMap-app-1*[*cat-Kan-cs-simps*]:
assumes *a* = $1_{\mathbb{N}}$
shows *LK-σ23* $\mathfrak{F}$ η' $\mathfrak{F}'(\!|NTMap|\!)(\!|a|\!)$ = $\mathfrak{F}'(\!|ArrMap|\!)(\!|0,\ 1_{\mathbb{N}}|\!)_{\bullet} \circ_{A\mathfrak{B}}\ \eta'(\!|NTMap|\!)(\!|0|\!)$
using *assms* **unfolding** *LK-σ23-components cat-ordinal-cs-simps cat-cs-simps* **by** *simp*

lemmas [*cat-Kan-cs-simps*] = *is-functor.LK-σ23-NTMap-app-1*

lemma *LK-σ23-NTMap-app-2*[*cat-Kan-cs-simps*]:
assumes *a* = $2_{\mathbb{N}}$
shows *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\!|NTMap|\!)(\!|a|\!)$ = $\eta'(\!|NTMap|\!)(\!|1_{\mathbb{N}}|\!)$
using *assms* **unfolding** *LK-σ23-components cat-ordinal-cs-simps* **by** *simp*

16.7.3 *LK-σ23* is a natural transformation

lemma *LK-σ23-is-ntcf*:
  assumes $\mathfrak{F}'$: *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $\mathfrak{T}$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
    and $\eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{K}23$: *cat-ordinal* $(2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
  shows *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'$: *LK23* $\mathfrak{T} \mapsto_{CF} \mathfrak{F}'$: *cat-ordinal* $(3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–

  interpret $\mathfrak{F}'$: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{F}'$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{T}$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{T}$ **by** (*rule assms*(*2*))
  interpret η': *is-ntcf* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› $\mathfrak{A}$ $\mathfrak{T}$ ‹$\mathfrak{F}' \circ_{CF} \mathfrak{K}23$›
    by (*rule assms*(*3*))

  interpret $\mathfrak{K}23$: *is-functor* α ‹*cat-ordinal* $(2_{\mathbb{N}})$› ‹*cat-ordinal* $(3_{\mathbb{N}})$› ‹$\mathfrak{K}23$›
    by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Kan-cs-intros*)
  interpret *LK23*: *is-functor* α ‹*cat-ordinal* $(3_{\mathbb{N}})$› $\mathfrak{A}$ ‹*LK23* $\mathfrak{T}$›
    by (*cs-concl* **cs-intro**: *cat-Kan-cs-intros cat-cs-intros*)

  show *?thesis*
  proof(*rule is-ntcfI'*)
    show *vfsequence* (*LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'$) **unfolding** *LK-σ23-def* **by** *simp*
    show *vcard* (*LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'$) = $5_{\mathbb{N}}$
      unfolding *LK-σ23-def* **by** (*simp-all add*: *nat-omega-simps*)
    show *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$: *LK23* $\mathfrak{T}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{A}} \mathfrak{F}'(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
      if $a \in_{\circ}$ *cat-ordinal* $(3_{\mathbb{N}})(\lfloor Obj \rfloor)$ **for** *a*
    proof–
      from *that* **consider** ‹$a = 0$› | ‹$a = 1_{\mathbb{N}}$› | ‹$a = 2_{\mathbb{N}}$›
        unfolding *cat-ordinal-cs-simps three* **by** *auto*
      from *this 0123* **show**
        LK-σ23 $\mathfrak{T}$ η' $\mathfrak{F}'(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$: *LK23* $\mathfrak{T}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{A}} \mathfrak{F}'(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
        by (*cases*, *use nothing* **in** ‹*simp-all only:*›)
          (
            cs-concl
              cs-simp: *cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps*
              cs-intro:
                cat-cs-intros
                cat-ordinal-cs-intros
                cat-Kan-cs-intros
                nat-omega-intros
)+
    qed
    show
      LK-σ23 $\mathfrak{T}$ η' $\mathfrak{F}'(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{A}}$ *LK23* $\mathfrak{T}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) =$
        $\mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{A}}$ *LK-σ23* $\mathfrak{T}$ η' $\mathfrak{F}'(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
      if $f : a \mapsto_{\textit{cat-ordinal}\ (3_{\mathbb{N}})} b$ **for** *a b f*
      using *that 0123*
      by (*elim cat-ordinal-3-is-arrE*, *use nothing* **in** ‹*simp-all only:*›)
        (
          cs-concl
            cs-simp:
              cat-cs-simps
              cat-ordinal-cs-simps
              $\mathfrak{F}'$*.cf-ArrMap-Comp*[*symmetric*]
              $\mathfrak{F}'$*.HomCod.cat-Comp-assoc*[*symmetric*]
              η'*.ntcf-Comp-commute*
              cat-Kan-cs-simps
            cs-intro: *cat-cs-intros cat-ordinal-cs-intros nat-omega-intros*

```
    )+
  qed
    (
      cs-concl
        cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros cat-Kan-cs-intros
    )+

qed

lemma LK-σ23-is-ntcf'[cat-Kan-cs-intros]:
  assumes 𝔉' : cat-ordinal (3ℕ) ↦↦Cα 𝔄
    and 𝔗 : cat-ordinal (2ℕ) ↦↦Cα 𝔄
    and η' : 𝔗 ↦CF 𝔉' ∘CF ℜ23 : cat-ordinal (2ℕ) ↦↦Cα 𝔄
    and 𝔊' = LK23 𝔗
    and ℌ' = 𝔉'
    and ℭ' = cat-ordinal (3ℕ)
  shows LK-σ23 𝔗 η' 𝔉' : 𝔊' ↦CF ℌ' : ℭ' ↦↦Cα 𝔄
  using assms(1–3) unfolding assms(4–6) by (rule LK-σ23-is-ntcf)
```

16.8 The left Kan extension along $\mathfrak{R}23$

```
lemma η23-is-cat-rKe:
  assumes 𝔗 : cat-ordinal (2ℕ) ↦↦Cα 𝔄
  shows ntcf-id 𝔗 :
    𝔗 ↦CF.lKeα LK23 𝔗 ∘CF ℜ23 : cat-ordinal (2ℕ) ↦C cat-ordinal (3ℕ) ↦C 𝔄
proof–

  interpret 𝔗: is-functor α ‹cat-ordinal (2ℕ)› 𝔄 𝔗 by (rule assms(1))
  interpret ℜ23: is-functor α ‹cat-ordinal (2ℕ)› ‹cat-ordinal (3ℕ)› ‹ℜ23›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros)
  interpret LK23: is-functor α ‹cat-ordinal (3ℕ)› 𝔄 ‹LK23 𝔗›
    by (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)

  show ?thesis
  proof(intro is-cat-lKeI')
    fix 𝔉' η' assume prems:
      𝔉' : cat-ordinal (3ℕ) ↦↦Cα 𝔄
      η' : 𝔗 ↦CF 𝔉' ∘CF ℜ23 : cat-ordinal (2ℕ) ↦↦Cα 𝔄

    interpret 𝔉': is-functor α ‹cat-ordinal (3ℕ)› 𝔄 𝔉' by (rule prems(1))
    interpret η': is-ntcf α ‹cat-ordinal (2ℕ)› 𝔄 𝔗 ‹𝔉' ∘CF ℜ23› η'
      by (rule prems(2))
    interpret LK-σ23: is-ntcf α ‹cat-ordinal (3ℕ)› 𝔄 ‹LK23 𝔗› 𝔉' ‹LK-σ23 𝔗 η' 𝔉'›
      by (intro LK-σ23-is-ntcf prems assms)

    show ∃!σ.
      σ : LK23 𝔗 ↦CF 𝔉' : cat-ordinal (3ℕ) ↦↦Cα 𝔄 ∧
      η' = σ ∘NTCF-CF ℜ23 ⋆NTCF ntcf-id 𝔗
    proof(intro ex1I conjI; (elim conjE)?)
      show LK-σ23 𝔗 η' 𝔉' : LK23 𝔗 ↦CF 𝔉' : cat-ordinal (3ℕ) ↦↦Cα 𝔄
        by (intro LK-σ23.is-ntcf-axioms)
      show η' = LK-σ23 𝔗 η' 𝔉' ∘NTCF-CF ℜ23 ⋆NTCF ntcf-id 𝔗
      proof(rule ntcf-eqI)
        show η' : 𝔗 ↦CF 𝔉' ∘CF ℜ23 : cat-ordinal (2ℕ) ↦↦Cα 𝔄
          by (intro prems)
        then have dom-lhs: 𝒟∘ (η'(|NTMap|)) = 2ℕ
          by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps)
        show rhs:
```

```
    LK-σ23 𝔗 η′ 𝔉′ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗 :
      𝔗 ↦_CF 𝔉′ ∘_CF ℜ23 : cat-ordinal (2_ℕ) ↦↦_Cα 𝔄
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps cat-cs-simps
          cs-intro: cat-Kan-cs-intros cat-cs-intros
      )
  then have dom-rhs:
    𝒟∘ ((LK-σ23 𝔗 η′ 𝔉′ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗)(⌊NTMap⌋)) = 2_ℕ
    by (cs-concl cs-simp: cat-ordinal-cs-simps cat-cs-simps)
  show η′(⌊NTMap⌋) = (LK-σ23 𝔗 η′ 𝔉′ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗)(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume prems: a ∈∘ 2_ℕ
    then consider ‹a = 0› | ‹a = 1_ℕ› unfolding two by auto
    then show
      η′(⌊NTMap⌋)(⌊a⌋) =
        (LK-σ23 𝔗 η′ 𝔉′ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗)(⌊NTMap⌋)(⌊a⌋)
      by (cases; use nothing in ‹simp-all only:›)
        (
          cs-concl
            cs-simp:
              omega-of-set
              cat-Kan-cs-simps
              cat-cs-simps
              cat-ordinal-cs-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros nat-omega-intros
        )+
  qed
    (
      use rhs in
        ‹cs-concl cs-shallow cs-intro: V-cs-intros cat-cs-intros›
    )+
qed simp-all

fix σ assume prems′:
  σ : LK23 𝔗 ↦_CF 𝔉′ : cat-ordinal (3_ℕ) ↦↦_Cα 𝔄
  η′ = σ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗

interpret σ: is-ntcf α ‹cat-ordinal (3_ℕ)› 𝔄 ‹LK23 𝔗› 𝔉′ σ
  by (rule prems′(1))

from prems′(2) have
  η′(⌊NTMap⌋)(⌊0⌋) = (σ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗)(⌊NTMap⌋)(⌊0⌋)
  by auto
then have [cat-cs-simps]: η′(⌊NTMap⌋)(⌊0⌋) = σ(⌊NTMap⌋)(⌊0⌋)
  by
    (
      cs-prems cs-shallow
        cs-simp: cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps
        cs-intro: cat-cs-intros nat-omega-intros
    )
from prems′(2) have
  η′(⌊NTMap⌋)(⌊1_ℕ⌋) = (σ ∘_NTCF-CF ℜ23 ⋆_NTCF ntcf-id 𝔗)(⌊NTMap⌋)(⌊1_ℕ⌋)
  by auto
then have [cat-cs-simps]: η′(⌊NTMap⌋)(⌊1_ℕ⌋) = σ(⌊NTMap⌋)(⌊2_ℕ⌋)
  by
    (
```

```
        cs-prems cs-shallow
          cs-simp:
            omega-of-set cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps
          cs-intro: cat-cs-intros nat-omega-intros
        )

    show σ = LK-σ23 𝔗 η′ 𝔉′
    proof(rule ntcf-eqI)

      show σ : LK23 𝔗 ↦_CF 𝔉′ : cat-ordinal (3_ℕ) ↦↦_Cα 𝔄
        by (rule prems′(1))
      then have dom-lhs: 𝒟∘ (σ⦇NTMap⦈) = 3_ℕ
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
      show LK-σ23 𝔗 η′ 𝔉′ : LK23 𝔗 ↦_CF 𝔉′ : cat-ordinal (3_ℕ) ↦↦_Cα 𝔄
        by (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros)
      then have dom-rhs: 𝒟∘ (LK-σ23 𝔗 η′ 𝔉′⦇NTMap⦈) = 3_ℕ
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
      from 0123 have 012: [0, 1_ℕ]∘ : 0 ↦_cat-ordinal (2_ℕ) 1_ℕ
        by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)
      from 0123 have 013: [0, 1_ℕ]∘ : 0 ↦_cat-ordinal (3_ℕ) 1_ℕ
        by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)
      from 0123 have 00: [0, 0]∘ = (cat-ordinal (2_ℕ))⦇CId⦈⦇0⦈
        by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps)
      from σ.ntcf-Comp-commute[OF 013] 013 0123
      have [symmetric, cat-Kan-cs-simps]:
        σ⦇NTMap⦈⦇1_ℕ⦈ = 𝔉′⦇ArrMap⦈⦇0, 1_ℕ⦈• ∘_A𝔄 σ⦇NTMap⦈⦇0⦈
        by
          (
            cs-prems
              cs-simp: cat-cs-simps cat-Kan-cs-simps 00 LK23-ArrMap-app-01
              cs-intro: cat-cs-intros cat-ordinal-cs-intros nat-omega-intros
          )

      show σ⦇NTMap⦈ = LK-σ23 𝔗 η′ 𝔉′⦇NTMap⦈
      proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
        fix a assume prems: a ∈∘ 3_ℕ
        then consider ‹a = 0› | ‹a = 1_ℕ› | ‹a = 2_ℕ› unfolding three by auto
        then show σ⦇NTMap⦈⦇a⦈ = LK-σ23 𝔗 η′ 𝔉′⦇NTMap⦈⦇a⦈
          by (cases; use nothing in ‹simp-all only:›)
            (
              cs-concl
                cs-simp: cat-ordinal-cs-simps cat-cs-simps cat-Kan-cs-simps
                cs-intro: cat-cs-intros
            )+
      qed auto
    qed simp-all

  qed

  qed (cs-concl cs-shallow cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)+

qed
```

16.9 Pointwise Kan extensions along $\mathfrak{K}23$

```
lemma ε23-is-cat-pw-rKe:
  assumes 𝔗 : cat-ordinal (2_ℕ) ↦↦_Cα 𝔄
  shows ntcf-id 𝔗 :
```

```
    RK23 𝔗 ∘_CF 𝔎23 ↦_CF.rKe.pwα 𝔗 :
    cat-ordinal (2_ℕ) ↦_C cat-ordinal (3_ℕ) ↦_C 𝔄
proof–

  interpret 𝔗: is-functor α ‹cat-ordinal (2_ℕ)› 𝔄 𝔗 by (rule assms(1))

  show ?thesis
  proof(intro is-cat-pw-rKeI ε23-is-cat-rKe[OF assms])

    fix a assume prems: a ∈₀ 𝔄(|Obj|)

    show
      ntcf-id 𝔗 :
        RK23 𝔗 ∘_CF 𝔎23 ↦_CF.rKeα 𝔗 :
        cat-ordinal (2_ℕ) ↦_C
        cat-ordinal (3_ℕ) ↦_C
        (Hom_O.Cα𝔄(a,−) : 𝔄 ↦↦_C cat-Set α)
    proof(intro is-cat-rKe-preservesI ε23-is-cat-rKe[OF assms])
      from prems show Hom_O.Cα𝔄(a,−) : 𝔄 ↦↦_Cα cat-Set α
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
      show Hom_O.Cα𝔄(a,−) ∘_CF−NTCF ntcf-id 𝔗 :
        (Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗) ∘_CF 𝔎23 ↦_CF.rKeα Hom_O.Cα𝔄(a,−) ∘_CF 𝔗 :
        cat-ordinal (2_ℕ) ↦_C cat-ordinal (3_ℕ) ↦_C cat-Set α
      proof(intro is-cat-rKeI′)
        show 𝔎23 : cat-ordinal (2_ℕ) ↦↦_Cα cat-ordinal (3_ℕ)
          by (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)
        from prems show
          Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗 : cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
          by (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros)
        from prems show
          Hom_O.Cα𝔄(a,−) ∘_CF−NTCF ntcf-id 𝔗 :
            Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗 ∘_CF 𝔎23 ↦_CF Hom_O.Cα𝔄(a,−) ∘_CF 𝔗 :
            cat-ordinal (2_ℕ) ↦↦_Cα cat-Set α
          by
            (
              cs-concl
                cs-simp: cat-cs-simps cat-Kan-cs-simps
                cs-intro: cat-cs-intros cat-Kan-cs-intros
            )

        fix 𝔊′ ε′ assume prems′:
          𝔊′ : cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
          ε′ :
            𝔊′ ∘_CF 𝔎23 ↦_CF Hom_O.Cα𝔄(a,−) ∘_CF 𝔗 :
            cat-ordinal (2_ℕ) ↦↦_Cα cat-Set α

        interpret 𝔊′: is-functor α ‹cat-ordinal (3_ℕ)› ‹cat-Set α› 𝔊′
          by (rule prems′(1))
        interpret ε′: is-ntcf
          α
          ‹cat-ordinal (2_ℕ)›
          ‹cat-Set α›
          ‹𝔊′ ∘_CF 𝔎23›
          ‹Hom_O.Cα𝔄(a,−) ∘_CF 𝔗›
          ε′
          by (rule prems′(2))

        show ∃!σ.
```

```
  σ :
    𝔊′ ↦_CF Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗 :
    cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α ∧
  ε′ = Hom_O.Cα𝔄(a,−) ∘_CF−NTCF ntcf-id 𝔗 ·_NTCF (σ ∘_NTCF−CF ℜ23)
proof(intro ex1I conjI; (elim conjE)?)
  have [cat-Kan-cs-simps]:
    Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗 = RK23 (Hom_O.Cα𝔄(a,−) ∘_CF 𝔗)
  proof(rule cf-eqI)
    from prems show lhs: Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗 :
      cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-Kan-cs-intros
        )
    from prems show rhs: RK23 (Hom_O.Cα𝔄(a,−) ∘_CF 𝔗) :
      cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-Kan-cs-intros
        )
    from lhs prems have ObjMap-dom-lhs:
      𝒟∘ ((Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗)(⦇ObjMap⦈)) = 3_ℕ
      by
        (
          cs-concl
            cs-simp: cat-ordinal-cs-simps cat-cs-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros
        )
    from rhs prems have ObjMap-dom-rhs:
      𝒟∘ ((RK23 (Hom_O.Cα𝔄(a,−) ∘_CF 𝔗))(⦇ObjMap⦈)) = 3_ℕ
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-ordinal-cs-simps cat-cs-simps
            cs-intro: cat-Kan-cs-intros
        )
    show
      (Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗)(⦇ObjMap⦈) =
        RK23 (Hom_O.Cα𝔄(a,−) ∘_CF 𝔗)(⦇ObjMap⦈)
    proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
      fix c assume prems′′: c ∈∘ 3_ℕ
      with 0123 consider ‹c = 0› | ‹c = 1_ℕ› | ‹c = 2_ℕ› by force
      from this prems prems′′ 0123 show
        (Hom_O.Cα𝔄(a,−) ∘_CF RK23 𝔗)(⦇ObjMap⦈)(⦇c⦈) =
          RK23 (Hom_O.Cα𝔄(a,−) ∘_CF 𝔗)(⦇ObjMap⦈)(⦇c⦈)
        by (cases; use nothing in ‹simp-all only:›)
          (
            cs-concl
              cs-simp:
                cat-ordinal-cs-simps
                cat-cs-simps
                cat-op-simps
                cat-Kan-cs-simps
              cs-intro: cat-Kan-cs-intros cat-cs-intros
```

```
    )+
  qed
    (
      use prems in ‹
        cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros
        ›
    )+
  from lhs prems have ArrMap-dom-lhs:
    𝒟∘ ((Hom_{O.Cα}𝔄(a,−) ∘_CF RK23 𝔗)(⌊ArrMap⌋)) =
      cat-ordinal (3_ℕ)(⌊Arr⌋)
    by
      (
        cs-concl
          cs-simp: cat-ordinal-cs-simps cat-cs-simps
          cs-intro: cat-Kan-cs-intros cat-cs-intros
      )
  from rhs prems have ArrMap-dom-rhs:
    𝒟∘ ((RK23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗))(⌊ArrMap⌋)) =
      cat-ordinal (3_ℕ)(⌊Arr⌋)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps cat-cs-simps
          cs-intro: cat-Kan-cs-intros
      )
  show
    (Hom_{O.Cα}𝔄(a,−) ∘_CF RK23 𝔗)(⌊ArrMap⌋) =
      RK23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗)(⌊ArrMap⌋)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix f assume prems'': f ∈∘ cat-ordinal (3_ℕ)(⌊Arr⌋)
    then obtain a' b' where f : a' ↦_{cat-ordinal (3_ℕ)} b' by auto
    from this 0123 prems show
      (Hom_{O.Cα}𝔄(a,−) ∘_CF RK23 𝔗)(⌊ArrMap⌋)(⌊f⌋) =
        RK23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗)(⌊ArrMap⌋)(⌊f⌋)
      by
        (
          elim cat-ordinal-3-is-arrE;
          use nothing in ‹simp-all only:›
        )
        (
          cs-concl
            cs-simp: cat-cs-simps cat-Kan-cs-simps cat-op-simps
            cs-intro:
              cat-ordinal-cs-intros
              cat-Kan-cs-intros
              cat-cs-intros
              nat-omega-intros
        )+
  qed
    (
      use prems in
        ‹cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros›
    )+
qed simp-all

show RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗) ε' 𝔊' :
  𝔊' ↦_CF Hom_{O.Cα}𝔄(a,−) ∘_CF RK23 𝔗 :
  cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
```

```
  by (intro RK-σ23-is-ntcf')
    (
      cs-concl cs-shallow
        cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
    )+
show ε' =
  Hom_{O.Cα}𝔄(a,−) ∘_{CF−NTCF}
  ntcf-id 𝔗 ⋆_{NTCF}
  (RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗) ε' 𝔊' ∘_{NTCF−CF} ℜ23)
proof(rule ntcf-eqI)
  show ε' :
    𝔊' ∘_{CF} ℜ23 ↦_{CF} Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗 :
    cat-ordinal (2_ℕ) ↦↦_{Cα} cat-Set α
    by (intro prems')
  then have dom-lhs: 𝒟∘ (ε'(|NTMap|)) = 2_ℕ
    by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps)
  from prems show
    Hom_{O.Cα}𝔄(a,−) ∘_{CF−NTCF}
      ntcf-id 𝔗 ⋆_{NTCF}
      (RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗) ε' 𝔊' ∘_{NTCF−CF} ℜ23) :
    𝔊' ∘_{CF} ℜ23 ↦_{CF} Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗 :
    cat-ordinal (2_ℕ) ↦↦_{Cα} cat-Set α
    by
      (
        cs-concl
          cs-simp: cat-Kan-cs-simps
          cs-intro: cat-Kan-cs-intros cat-cs-intros
      )
  then have dom-rhs:
    𝒟∘
      (
        (Hom_{O.Cα}𝔄(a,−) ∘_{CF−NTCF}
        ntcf-id 𝔗 ⋆_{NTCF}
        (RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗) ε' 𝔊' ∘_{NTCF−CF} ℜ23)
      )(|NTMap|)) = 2_ℕ
    by (cs-concl cs-simp: cat-ordinal-cs-simps cat-cs-simps)
  show ε'(|NTMap|) =
    (
      Hom_{O.Cα}𝔄(a,−) ∘_{CF−NTCF}
      ntcf-id 𝔗 ⋆_{NTCF}
      (RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗) ε' 𝔊' ∘_{NTCF−CF} ℜ23)
    )(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix c assume prems'': c ∈∘ 2_ℕ
    then consider ‹c = 0› | ‹c = 1_ℕ› unfolding two by auto
    from this prems 0123 show ε'(|NTMap|)(|c|) =
      (
        Hom_{O.Cα}𝔄(a,−) ∘_{CF−NTCF}
        ntcf-id 𝔗 ⋆_{NTCF}
        (RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_{CF} 𝔗) ε' 𝔊' ∘_{NTCF−CF} ℜ23)
      )(|NTMap|)(|c|)
      by (cases; use nothing in ‹simp-all only:›)
        (
          cs-concl
            cs-simp:
              cat-Kan-cs-simps
              cat-ordinal-cs-simps
              cat-cs-simps
```

```
            cat-op-simps
            RK-σ23-NTMap-app-0
            cat-Set-components(1)
          cs-intro:
            cat-Kan-cs-intros
            cat-cs-intros
            cat-prod-cs-intros
            𝔗.HomCod.cat-Hom-in-Vset
      )+
  qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)+

qed simp-all

fix σ assume prems'':
  σ :
    𝔊' ↦CF Hom_O.Cα 𝔄(a,−) ∘CF RK23 𝔗 :
    cat-ordinal (3ℕ) ↦↦Cα cat-Set α
  ε' = Hom_O.Cα 𝔄(a,−) ∘CF−NTCF ntcf-id 𝔗 ⋆NTCF (σ ∘NTCF−CF ℜ23)

interpret σ: is-ntcf
  α ‹cat-ordinal (3ℕ)› ‹cat-Set α› 𝔊' ‹Hom_O.Cα 𝔄(a,−) ∘CF RK23 𝔗› σ
  by (rule prems''(1))

from prems''(2) have ε'(⌊NTMap⌋)(⌊0⌋) =
  (Hom_O.Cα 𝔄(a,−) ∘CF−NTCF ntcf-id 𝔗 ⋆NTCF (σ ∘NTCF−CF ℜ23))(⌊NTMap⌋)(⌊0⌋)
  by auto
from this prems 0123 have ε'-NTMap-app-0: ε'(⌊NTMap⌋)(⌊0⌋) = σ(⌊NTMap⌋)(⌊0⌋)
  by
    (
      cs-prems
        cs-simp:
          cat-ordinal-cs-simps
          cat-cs-simps
          cat-Kan-cs-simps
          cat-op-simps
          ℜ23-ObjMap-app-0
          cat-Set-components(1)
        cs-intro:
          cat-Kan-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          𝔗.HomCod.cat-Hom-in-Vset
    )
from 0123 have 01: [0, 1ℕ]∘ : 0 ↦cat-ordinal (2ℕ) 1ℕ
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-ordinal-cs-intros nat-omega-intros
    )
from prems''(2) have
  ε'(⌊NTMap⌋)(⌊1ℕ⌋) =
    (
      Hom_O.Cα 𝔄(a,−) ∘CF−NTCF ntcf-id 𝔗 ⋆NTCF (σ ∘NTCF−CF ℜ23)
    )(⌊NTMap⌋)(⌊1ℕ⌋)
  by auto
from this prems 0123 have ε'-NTMap-app-1:
  ε'(⌊NTMap⌋)(⌊1ℕ⌋) = σ(⌊NTMap⌋)(⌊2ℕ⌋)
```

```
  by
    (
      cs-prems
        cs-simp:
          cat-ordinal-cs-simps
          cat-cs-simps
          cat-Kan-cs-simps
          cat-op-simps
          ℜ23-ObjMap-app-1
          cat-Set-components(1)
        cs-intro:
          cat-Kan-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          𝔗.HomCod.cat-Hom-in-Vset
    )

from 0123 have 012: [0, 1ℕ]∘ : 0 ↦_cat-ordinal (2ℕ) 1ℕ
  by
    (
      cs-concl cs-intro:
        cat-ordinal-cs-intros nat-omega-intros
    )
from 0123 have 013: [0, 1ℕ]∘ : 0 ↦_cat-ordinal (3ℕ) 1ℕ
  by
    (
      cs-concl cs-intro:
        cat-ordinal-cs-intros nat-omega-intros
    )
from 0123 have 123: [1ℕ, 2ℕ]∘ : 1ℕ ↦_cat-ordinal (3ℕ) 2ℕ
  by
    (
      cs-concl cs-intro:
        cat-ordinal-cs-intros nat-omega-intros
    )
from 0123 have 11: [1ℕ, 1ℕ]∘ = (cat-ordinal (2ℕ))(‖CId‖)(‖1ℕ‖)
  by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps)

from σ.ntcf-Comp-commute[OF 123] prems 012 013
have [cat-Kan-cs-simps]:
  ε′(‖NTMap‖)(‖1ℕ‖) ∘_A cat-Set α 𝔊′(‖ArrMap‖)(‖1ℕ, 2ℕ‖)• = σ(‖NTMap‖)(‖1ℕ‖)
  by
    (
      cs-prems
        cs-simp:
          cat-cs-simps
          cat-Kan-cs-simps
          ε′-NTMap-app-1[symmetric]
          is-functor.cf-ObjMap-CId
          RK23-ArrMap-app-12
          11
        cs-intro: cat-cs-intros nat-omega-intros
    )

show σ = RK-σ23 (Hom_O.Cα𝔄(a,–) ∘_CF 𝔗) ε′ 𝔊′
proof(rule ntcf-eqI)

  show σ: σ :
```

```
      𝔊′ ↦_CF Hom_{O.Cα}𝔄(a,−) ∘_CF RK23 𝔗 :
      cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
      by (rule prems′′(1))
    then have dom-lhs: 𝒟∘ (σ⟦NTMap⟧) = 3_ℕ
      by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps)
    show RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗) ε′ 𝔊′ :
      𝔊′ ↦_CF Hom_{O.Cα}𝔄(a,−) ∘_CF RK23 𝔗 :
      cat-ordinal (3_ℕ) ↦↦_Cα cat-Set α
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-Kan-cs-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros
        )
    then have dom-rhs:
      𝒟∘ (RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗) ε′ 𝔊′⟦NTMap⟧) = 3_ℕ
      by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps)
    show σ⟦NTMap⟧ = RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗) ε′ 𝔊′⟦NTMap⟧
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix c assume c ∈∘ 3_ℕ
      then consider ‹c = 0› | ‹c = 1_ℕ› | ‹c = 2_ℕ›
        unfolding three by auto
      from this 0123 show
        σ⟦NTMap⟧⟦c⟧ = RK-σ23 (Hom_{O.Cα}𝔄(a,−) ∘_CF 𝔗) ε′ 𝔊′⟦NTMap⟧⟦c⟧
        by (cases; use nothing in ‹simp-all only:›)
          (
            cs-concl cs-simp:
              cat-Kan-cs-simps ε′-NTMap-app-1 ε′-NTMap-app-0
          )+
    qed (cs-concl cs-intro: cat-Kan-cs-intros V-cs-intros)+

  qed simp-all

 qed

qed

qed

qed

lemma η23-is-cat-pw-lKe:
  assumes 𝔗 : cat-ordinal (2_ℕ) ↦↦_Cα 𝔄
  shows ntcf-id 𝔗 :
    𝔗 ↦_{CF.lKe.pwα} LK23 𝔗 ∘_CF ℜ23 :
    cat-ordinal (2_ℕ) ↦_C cat-ordinal (3_ℕ) ↦_C 𝔄
proof–

  interpret 𝔗: is-functor α ‹cat-ordinal (2_ℕ)› 𝔄 𝔗 by (rule assms(1))

  from ord-of-nat-ω interpret cat-ordinal-3: finite-category α ‹cat-ordinal (3_ℕ)›
    by (cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros)

  from 0123 have 002: [0, 0]∘ : 0 ↦_{cat-ordinal (2_ℕ)} 0
    by
      (
```

```
    cs-concl cs-shallow
      cs-simp: cat-ordinal-cs-simps cs-intro: cat-cs-intros
  )

show ?thesis
proof(intro is-cat-pw-lKeI η23-is-cat-rKe assms, unfold cat-op-simps)
  fix a assume prems: a ∈∘ 𝔄(|Obj|)
  show
    op-ntcf (ntcf-id 𝔗) :
      op-cf (LK23 𝔗) ∘CF op-cf ℜ23 ↦CF.rKeα op-cf 𝔗 :
      op-cat (cat-ordinal (2ℕ)) ↦C op-cat (cat-ordinal (3ℕ)) ↦C
      (HomO.Cα𝔄(−,a) : op-cat 𝔄 ↦↦C cat-Set α)
  proof(intro is-cat-rKe-preservesI)
    show
      op-ntcf (ntcf-id 𝔗) :
        op-cf (LK23 𝔗) ∘CF op-cf ℜ23 ↦CF.rKeα op-cf 𝔗 :
        op-cat (cat-ordinal (2ℕ)) ↦C op-cat (cat-ordinal (3ℕ)) ↦C op-cat 𝔄
    proof(cs-intro-step cat-op-intros)
      show ntcf-id 𝔗 :
        𝔗 ↦CF.lKeα LK23 𝔗 ∘CF ℜ23 :
        cat-ordinal (2ℕ) ↦C cat-ordinal (3ℕ) ↦C 𝔄
        by (intro η23-is-cat-rKe assms)
    qed simp-all
    from prems show HomO.Cα𝔄(−,a) : op-cat 𝔄 ↦↦Cα cat-Set α
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)

    have
      op-cf HomO.Cα𝔄(−,a) ∘CF−NTCF ntcf-id 𝔗 :
        op-cf HomO.Cα𝔄(−,a) ∘CF 𝔗 ↦CF.lKeα
        (op-cf HomO.Cα𝔄(−,a) ∘CF LK23 𝔗) ∘CF ℜ23 :
        cat-ordinal (2ℕ) ↦C cat-ordinal (3ℕ) ↦C op-cat (cat-Set α)
    proof(intro is-cat-lKeI′)
      show ℜ23 : cat-ordinal (2ℕ) ↦↦Cα cat-ordinal (3ℕ)
        by (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)
      from prems show op-cf HomO.Cα𝔄(−,a) ∘CF LK23 𝔗 :
        cat-ordinal (3ℕ) ↦↦Cα op-cat (cat-Set α)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-op-simps
              cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
          )

      from prems show
        op-cf HomO.Cα𝔄(−,a) ∘CF−NTCF ntcf-id 𝔗 :
          op-cf HomO.Cα𝔄(−,a) ∘CF 𝔗 ↦CF
          op-cf HomO.Cα𝔄(−,a) ∘CF LK23 𝔗 ∘CF ℜ23 :
          cat-ordinal (2ℕ) ↦↦Cα op-cat (cat-Set α)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-Kan-cs-simps cat-op-simps
              cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
          )

      fix 𝔉′ η′ assume prems′:
        𝔉′ : cat-ordinal (3ℕ) ↦↦Cα op-cat (cat-Set α)
        η′ :
```

```
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗 ↦_{CF} 𝔉′ ∘_{CF} ℜ23 :
    cat-ordinal (2_ℕ) ↦↦_{Cα} op-cat (cat-Set α)

interpret 𝔉′: is-functor α ‹cat-ordinal (3_ℕ)› ‹op-cat (cat-Set α)› 𝔉′
  by (rule prems′(1))
interpret η′: is-ntcf
  α
  ‹cat-ordinal (2_ℕ)›
  ‹op-cat (cat-Set α)›
  ‹op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗›
  ‹𝔉′ ∘_{CF} ℜ23›
  η′
  by (rule prems′(2))
note [unfolded cat-op-simps, cat-cs-intros] =
  η′.ntcf-NTMap-is-arr′
  𝔉′.cf-ArrMap-is-arr′
show
  ∃!σ.
    σ :
      op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗 ↦_{CF} 𝔉′ :
      cat-ordinal (3_ℕ) ↦↦_{Cα} op-cat (cat-Set α) ∧
    η′ = σ ∘_{NTCF−CF} ℜ23 ⋆_{NTCF} (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF−NTCF} ntcf-id 𝔗)
proof(intro ex1I conjI; (elim conjE)?)
  have [cat-Kan-cs-simps]:
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗 =
      LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗)
  proof(rule cf-eqI)
    from prems show lhs: op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗 :
      cat-ordinal (3_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
      by
        (
          cs-concl
            cs-simp: cat-op-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
        )
    from prems show rhs: LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) :
      cat-ordinal (3_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
      by (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)
    from lhs prems have ObjMap-dom-lhs:
      𝒟∘ ((op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗)(‖ObjMap‖)) = 3_ℕ
      by
        (
          cs-concl
            cs-simp: cat-ordinal-cs-simps cat-cs-simps cat-op-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros
        )
    from rhs prems have ObjMap-dom-rhs:
      𝒟∘ (LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗)(‖ObjMap‖)) = 3_ℕ
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-ordinal-cs-simps cat-cs-simps
        )
    show
      (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗)(‖ObjMap‖) =
        LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗)(‖ObjMap‖)
    proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
     fix c assume prems′′: c ∈∘ 3_ℕ
```

```
 then consider ‹c = 0› | ‹c = 1ₙ› | ‹c = 2ₙ›
  unfolding three by auto
 from this prems 0123 show
   (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF LK23 𝔗)(|ObjMap|)(|c|) =
     LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗)(|ObjMap|)(|c|)
   by (cases; use nothing in ‹simp-all only:›)
     (
       cs-concl
         cs-simp:
           cat-ordinal-cs-simps
           cat-Kan-cs-simps
           cat-cs-simps
           cat-op-simps
         cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
     )+
qed
 (
   use prems in
     ‹
       cs-concl
         cs-simp: cat-op-simps
         cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
     ›
 )+

from lhs prems have ArrMap-dom-lhs:
  𝒟∘ ((op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF LK23 𝔗)(|ArrMap|)) =
    cat-ordinal (3ₙ)(|Arr|)
  by
    (
      cs-concl
        cs-simp: cat-ordinal-cs-simps cat-cs-simps cat-op-simps
        cs-intro: cat-Kan-cs-intros cat-cs-intros
    )
from rhs prems have ArrMap-dom-rhs:
  𝒟∘ (LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗)(|ArrMap|)) =
    cat-ordinal (3ₙ)(|Arr|)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)

show
  (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF LK23 𝔗)(|ArrMap|) =
    LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗)(|ArrMap|)
proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
  fix f assume f ∈∘ cat-ordinal (3ₙ)(|Arr|)
  then obtain a′ b′ where f: f : a′ ↦_{cat-ordinal (3ₙ)} b′
    by auto
  from f prems 0123 002 show
    (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF LK23 𝔗)(|ArrMap|)(|f|) =
      LK23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗)(|ArrMap|)(|f|)
    by (elim cat-ordinal-3-is-arrE, (simp-all only:)?)
      (
        cs-concl
          cs-simp: cat-cs-simps cat-Kan-cs-simps cat-op-simps
          cs-intro:
            cat-ordinal-cs-intros
            cat-Kan-cs-intros
            cat-cs-intros
            cat-op-intros
```

```
            nat-omega-intros
        )+
  qed
    (
      use prems in
        ‹
          cs-concl
            cs-simp: cat-op-simps
            cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros›
    )+

qed simp-all

show LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′ :
  op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗 ↦_{CF} 𝔉′ :
  cat-ordinal (3_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-Kan-cs-simps cat-op-simps
        cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
    )

show η′ =
  LK-σ23
    (
      op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′ ∘_{NTCF−CF}
      ℜ23 ∘_{NTCF}
      (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF−NTCF} ntcf-id 𝔗
    )
proof(rule ntcf-eqI)
  show lhs: η′ :
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗 ↦_{CF} 𝔉′ ∘_{CF} ℜ23 :
    cat-ordinal (2_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
    by (rule prems′(2))
  from lhs have 𝒟∘ (η′(|NTMap|)) = cat-ordinal (2_ℕ)(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from prems show rhs:
    LK-σ23
      (
        op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′ ∘_{NTCF−CF}
        ℜ23 ∘_{NTCF}
        (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF−NTCF} ntcf-id 𝔗
      ) :
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗 ↦_{CF} 𝔉′ ∘_{CF} ℜ23 :
    cat-ordinal (2_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
    by
      (
        cs-concl
          cs-simp: cat-Kan-cs-simps cat-op-simps
          cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
      )
  from lhs have dom-lhs: 𝒟∘ (η′(|NTMap|)) = 2_ℕ
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps cat-cs-simps
      )
```

```
  from rhs have dom-rhs: 𝒟∘ ((LK-σ23
    (
      op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗) η′ 𝔉′ ∘_NTCF−CF
      ℜ23 ⋆_NTCF
      (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF−NTCF ntcf-id 𝔗
    ))(⌊NTMap⌋)) = 2_ℕ
    by (cs-concl cs-simp: cat-ordinal-cs-simps cat-cs-simps)
  show
    η′(⌊NTMap⌋) =
      (
        LK-σ23
          (
            op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗) η′ 𝔉′ ∘_NTCF−CF
            ℜ23 ⋆_NTCF
            (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF−NTCF ntcf-id 𝔗
          )
      )(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-ordinal-cs-simps)
      fix c assume c ∈∘ 2_ℕ
      then consider ‹c = 0› | ‹c = 1_ℕ› unfolding two by auto
      from this prems 0123 show
        η′(⌊NTMap⌋)(⌊c⌋) =
          (
            LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF 𝔗) η′ 𝔉′ ∘_NTCF−CF
            ℜ23 ⋆_NTCF (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF−NTCF ntcf-id 𝔗)
          )(⌊NTMap⌋)(⌊c⌋)
        by (cases, use nothing in ‹simp-all only:›)
          (
            cs-concl
              cs-simp:
                cat-ordinal-cs-simps
                cat-Kan-cs-simps
                cat-cs-simps
                cat-op-simps
                ℜ23-ObjMap-app-1
                ℜ23-ObjMap-app-0
                LK-σ23-NTMap-app-0
                cat-Set-components(1)
              cs-intro:
                cat-Kan-cs-intros
                cat-cs-intros
                cat-prod-cs-intros
                cat-op-intros
                𝔗.HomCod.cat-Hom-in-Vset
          )+
    qed (cs-concl cs-intro: V-cs-intros cat-cs-intros)+
qed simp-all

fix σ assume prems′′:
  σ :
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF LK23 𝔗 ↦_CF 𝔉′ :
    cat-ordinal (3_ℕ) ↦↦_Cα op-cat (cat-Set α)
  η′ = σ ∘_NTCF−CF ℜ23 ⋆_NTCF (op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF−NTCF ntcf-id 𝔗)

interpret σ: is-ntcf
  α
  ‹cat-ordinal (3_ℕ)› ‹op-cat (cat-Set α)›
  ‹op-cf Hom_{O.Cα}𝔄(−,a) ∘_CF LK23 𝔗›
```

```
  𝔉′
  σ
  by (rule prems''(1))

note [cat-Kan-cs-intros] = σ.ntcf-NTMap-is-arr'[unfolded cat-op-simps]

from prems''(2) have
  η'(|NTMap|)(|0|) =
    (
      σ ∘_NTCF-CF
      ℜ23 ·_NTCF
      (op-cf Hom_O.Cα𝔄(−,a) ∘_CF-NTCF ntcf-id 𝔗)
    )(|NTMap|)(|0|)
  by simp
from this prems 0123 have η'-NTMap-app-0: η'(|NTMap|)(|0|) = σ(|NTMap|)(|0|)
  by
    (
      cs-prems
        cs-simp:
          cat-ordinal-cs-simps
          cat-Kan-cs-simps
          cat-cs-simps
          cat-op-simps
          cat-Set-components(1)
        cs-intro:
          cat-Kan-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          cat-op-intros
          𝔗.HomCod.cat-Hom-in-Vset
    )

from prems''(2) have
  η'(|NTMap|)(|1_ℕ|) =
    (
      σ ∘_NTCF-CF
      ℜ23 ·_NTCF
      (op-cf Hom_O.Cα𝔄(−,a) ∘_CF-NTCF ntcf-id 𝔗)
    )(|NTMap|)(|1_ℕ|)
  by simp
from this prems 0123 have η'-NTMap-app-1: η'(|NTMap|)(|1_ℕ|) = σ(|NTMap|)(|2_ℕ|)
  by
    (
      cs-prems
        cs-simp:
          cat-ordinal-cs-simps
          cat-Kan-cs-simps
          cat-cs-simps
          cat-op-simps
          cat-Set-components(1)
        cs-intro:
          cat-Kan-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          cat-op-intros
          𝔗.HomCod.cat-Hom-in-Vset
    )+
```

```
from 0123 have 013: [0, 1_ℕ]_∘ : 0 ↦_{cat-ordinal (3_ℕ)} 1_ℕ
  by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)
from 0123 have 00: [0, 0]_∘ = (cat-ordinal (2_ℕ))(|CId|)(|0|)
  by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps)

from σ.ntcf-Comp-commute[OF 013] prems 0123 013
have [cat-Kan-cs-simps]:
  σ(|NTMap|)(|1_ℕ|) = η′(|NTMap|)(|0|) ∘_{A cat-Set α} 𝔉′(|ArrMap|)(|0, 1_ℕ|)•
  by
    (
      cs-prems
        cs-simp:
          cat-ordinal-cs-simps
          cat-Kan-cs-simps
          cat-cs-simps
          cat-op-simps
          LK23-ArrMap-app-01
        cs-intro:
          cat-ordinal-cs-intros
          cat-Kan-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          cat-op-intros
          nat-omega-intros
        cs-simp: 00 η′-NTMap-app-0[symmetric]
    )

show σ = LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′
proof(rule ntcf-eqI)
  show lhs: σ :
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗 ↦_{CF} 𝔉′ :
    cat-ordinal (3_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
    by (rule prems′′(1))
  show rhs: LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′ :
    op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} LK23 𝔗 ↦_{CF} 𝔉′ :
    cat-ordinal (3_ℕ) ↦↦_{Cα} op-cat (cat-Set α)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Kan-cs-simps
          cs-intro: cat-Kan-cs-intros cat-cs-intros
      )
  from lhs have dom-lhs: 𝒟_∘ (σ(|NTMap|)) = 3_ℕ
    by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps)
  from rhs have dom-rhs:
    𝒟_∘ (LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′(|NTMap|)) = 3_ℕ
    by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps)

  show σ(|NTMap|) = LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix c assume c ∈_∘ 3_ℕ
    then consider ‹c = 0› | ‹c = 1_ℕ› | ‹c = 2_ℕ›
      unfolding three by auto
    from this 0123 show
      σ(|NTMap|)(|c|) =
        LK-σ23 (op-cf Hom_{O.Cα}𝔄(−,a) ∘_{CF} 𝔗) η′ 𝔉′(|NTMap|)(|c|)
      by (cases, use nothing in ‹simp-all only:›)
        (
```

```
        cs-concl
          cs-simp:
            cat-ordinal-cs-simps
            cat-cs-simps
            cat-Kan-cs-simps
            cat-op-simps
            η′-NTMap-app-0
            LK-σ23-NTMap-app-0
            η′-NTMap-app-1
          cs-intro:
            cat-ordinal-cs-intros
            cat-Kan-cs-intros
            cat-cs-intros
            cat-op-intros
            nat-omega-intros
        )+
    qed (cs-concl cs-intro: cat-Kan-cs-intros V-cs-intros)+

  qed simp-all

 qed

qed

then have
  op-ntcf (Hom_{O.Cα}𝔄(−,a) ∘_{CF−NTCF} op-ntcf (ntcf-id 𝔗)) :
    op-cf (Hom_{O.Cα}𝔄(−,a) ∘_{CF} op-cf 𝔗) ↦_{CF.lKeα}
    op-cf ((Hom_{O.Cα}𝔄(−,a) ∘_{CF} op-cf (LK23 𝔗))) ∘_{CF} op-cf (op-cf ℜ23) :
    op-cat (op-cat (cat-ordinal (2_ℕ))) ↦_C
    op-cat (op-cat (cat-ordinal (3_ℕ))) ↦_C
    op-cat (cat-Set α)
  by
    (
      cs-concl
        cs-simp: cat-op-simps
        cs-intro: cat-cs-intros cat-Kan-cs-intros cat-op-intros
    )
from is-cat-lKe.is-cat-rKe-op[OF this] prems show
  Hom_{O.Cα}𝔄(−,a) ∘_{CF−NTCF} op-ntcf (ntcf-id 𝔗) :
    (Hom_{O.Cα}𝔄(−,a) ∘_{CF} op-cf (LK23 𝔗)) ∘_{CF} op-cf ℜ23 ↦_{CF.rKeα}
    Hom_{O.Cα}𝔄(−,a) ∘_{CF} op-cf 𝔗 :
    op-cat (cat-ordinal (2_ℕ)) ↦_C
    op-cat (cat-ordinal (3_ℕ)) ↦_C
    cat-Set α
  by
    (
      cs-prems
        cs-simp: cat-op-simps
        cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
    )

 qed

qed

qed
```

References

[1] nLab. URL https://ncatlab.org/nlab/show/HomePage.

[2] Wikipedia, 2001. URL https://www.wikipedia.org/.

[3] S. Awodey. *Category Theory*. Oxford University Press, Oxford, UK, 2010. ISBN 978-0-19-958736-0.

[4] P. Bodo. *Categories and Functors*. Academic Press, New York, 1970.

[5] F. Haftmann. Sketch-and-Explore, 2021. URL https://isabelle.in.tum.de/library/HOL/HOL-ex/Sketch_and_Explore.html.

[6] T. W. Hungerford. *Algebra*. Springer, New York, 2003. ISBN 978-0-387-90518-1.

[7] D. M. Kan. Adjoint Functors. *Transactions of the American Mathematical Society*, 87(2): 294–329, 1958.

[8] M. C. Lehner. "All Concepts are Kan Extensions": Kan Extensions as the Most Universal of the Universal Constructions. Bachelor of Arts Thesis, Harvard College, Cambridge, MA, 2014.

[9] S. Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer, New York, 2 edition, 2010. ISBN 978-1-4419-3123-8.

[10] M. Milehins. Category Theory for ZFC in HOL II: Elementary Theory of 1-Categories. *Archive of Formal Proofs*, 2021.

[11] S. Obua. Partizan Games in Isabelle/HOLZF. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *ICTAC 2006*, volume 4281, pages 272–286. Springer, Berlin, 2006. ISBN 978-3-540-48815-6.

[12] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986. doi: 10.1016/0743-1066(86)90015-4.

[13] L. C. Paulson. Zermelo Fraenkel Set Theory in Higher-Order Logic. *Archive of Formal Proofs*, 2019.

[14] E. Riehl. *Category Theory in Context*. Emily Riehl, 2016.